(2)

RL-TR-91-27
Final Technical Report
March 1991

ROME LABORATORY

ADAPTIVE INTERFACES

SRI International

Philip Cohen

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

APPROVED: Sharon M. Walter

SHARON M. WALTER
Project Engineer

APPROVED: Raymond P. Urtz Jr

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Ronald Raposo

RONALD RAPOSO
Directorate of Plans & Programs

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 1991	3. REPORT TYPE AND DATES COVERED Final Feb 89 - Apr 90

4. TITLE AND SUBTITLE ADAPTIVE INTERFACES	5. FUNDING NUMBERS C - F30602-87-D-0094, Task 7 PE - 62702F PR - 5581 TA - QC WU - 07
6. AUTHOR(S) Philip Cohen	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Avenue Menlo Park CA 94025	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (COES) Griffiss AFB NY 13441-5700	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-91-27

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Sharon M. Walter/COES/(315) 330-7650

Prime Contractor: IIT Research Institute (IITRI)

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Shoptalk system served in this task, as a testbed for integrating Natural Language and graphics techniques. The goals were to: 1) develop a uniform semantics of pointing for devices such as a mouse, 2) develop a multi-modal interface methodology that includes the ability to use Natural Language to describe objects and time periods and the ability to use graphical techniques to manipulate context, and 3) conduct a feasibility demonstration of a prototype adaptive interface.

14. SUBJECT TERMS Man-Machine Interface, Natural Language Processing, Interface, Shoptalk	15. NUMBER OF PAGES 60
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

FOREWORD

This is the Final Technical Report, CDRL No. G003, for Task 7 under contract F30602-87-D-0094. This contract is with IIT Research Institute (IITRI) and is sponsored by the Rome Air Development Center. The work was performed by SRI International under subcontract R009406 with IITRI providing management support.

Contents

List of Figures

Chapter 1

Summary

SRI International (SRI) is pleased to present to Illinois Institute of Technology Research Institute (IITRI) and Rome Air Development Center (RADC) this Final Report for Project 7288, Adaptive Interfaces, Task A. This report is divided into three chapters and an appendix. Chapter 1 summarizes the goals and accomplishments of the Task. Chapter 2 presents technical accomplishments in integrated interfaces, and Chapter 3 presents technical accomplishments in a next generation natural language parser, grammar, and grammar development environment. Finally, the appendix presents background material useful for understanding our approach.

1.1 Objectives

The objectives of this effort were to:

1. Develop a uniform semantics of pointing (e.g., mouse or light pen). This semantic representation should be used to incorporate objects pointed-at, either from map-like representations or display forms, into natural language expressions.

2. Develop a multi-modal interface methodology to command-and-control style applications that integrates Natural Language and Direct Manipulation technologies. The interface shall include at least the following functions:

   (a) The ability to use Natural Language to describe objects and time periods that shall be included in actions selected from menus.

   (b) The ability to use graphical techniques to manipulate context. These techniques will allow users to ask follow-up questions that incorporate the selection of elements of display forms comprising answers to previous queries.

3. Conduct a feasibility demonstration of a prototype adaptive person-system interface embodying the semantics from paragraph 1 and the development from paragraph 2. The demonstration shall be held at SRI International.

4. Provide and deliver a video tape documenting the demonstration conducted in paragraph 3. This videotape will be prepared in both 3/4-inch UMATIC and 1/2-inch VHS format according to the best commercial practices. A narration describing the demonstration shall be included on the tape. One (1) copy of each format tape shall be delivered to the Government.

5. Oral presentations shall be held at times and places in accordance with the Task Schedule.

6. The first status report will be provided 40 days after the start of this task (TASK CLIN 0001AG), and monthly thereafter via letter of transmittal. A final technical report will be submitted thirty (30) days after the completion of this task (TASK CLIN 0001AG).

1.2 Accomplishments

The aforementioned semantics and methodology have been developed, and are embodied in Shoptalk, our factory command-and-control system (see accompanying videotape). The feasibility demonstration was given at SRI International 3/8/90 to the Project Monitor from RADC. In addition to these accomplishments, and in preparation for integrating a spoken language capability into the integrated interface, SRI International developed a new keyboard parsing algorithm to replace the existing one in Shoptalk. This algorithm is a variant of the one being developed by SRI for the spoken language work funded by DARPA. In order to develop a version of Shoptalk that used this algorithm, we constructed a new grammar, semantics, grammar-development environment, and logical-form conversion routine. The collection of routines developed on this project are termed the New Language Engine (NLE). The following sections discuss these accomplishments in greater detail.

Chapter 2

Technical Accomplishments: Integrated Interfaces

2.1 Background: Shoptalk

Shoptalk is a prototype factory information system being developed by SRI to support situation assessment, logistical planning, and scenario evaluation. Factories serve as fertile environments for the study of a variety of issues related to intelligent interfaces and command-and-control because factory decision-makers often need to plan their actions for altering an ongoing process, one whose trajectory is only partially predictable. Also, such users need to contend with unforseen circumstances, as well as the normal circumstances involving the movement of objects to new locations where new actions are performed. The temporal nature of operations planning stresses the representation of time in the user interface, and makes interacting with simulation, planning, and scheduling systems imperative.

Shoptalk is designed to help factory personnel perform tasks such as quality assurance monitoring, work-in-progress tracking, and production scheduling, which involve inquiry about the factory's past, present, and future, respectively. The system allows users to query databases on the current state and recent history of the factory with a combination of English and graphical interaction techniques, and to examine alternative factory scenarios by running a discrete event simulator. The system features an integrated interface that permits intermixing natural-language queries and descriptions with mouse-pointing, menu selection, and graphical output. The current version of Shoptalk demonstrates the application of the technology to semiconductor and printed circuit-board manufacturing, but the basic system is equally applicable to a wide variety of command-and-control domains.

2.2 Integrated Interfaces

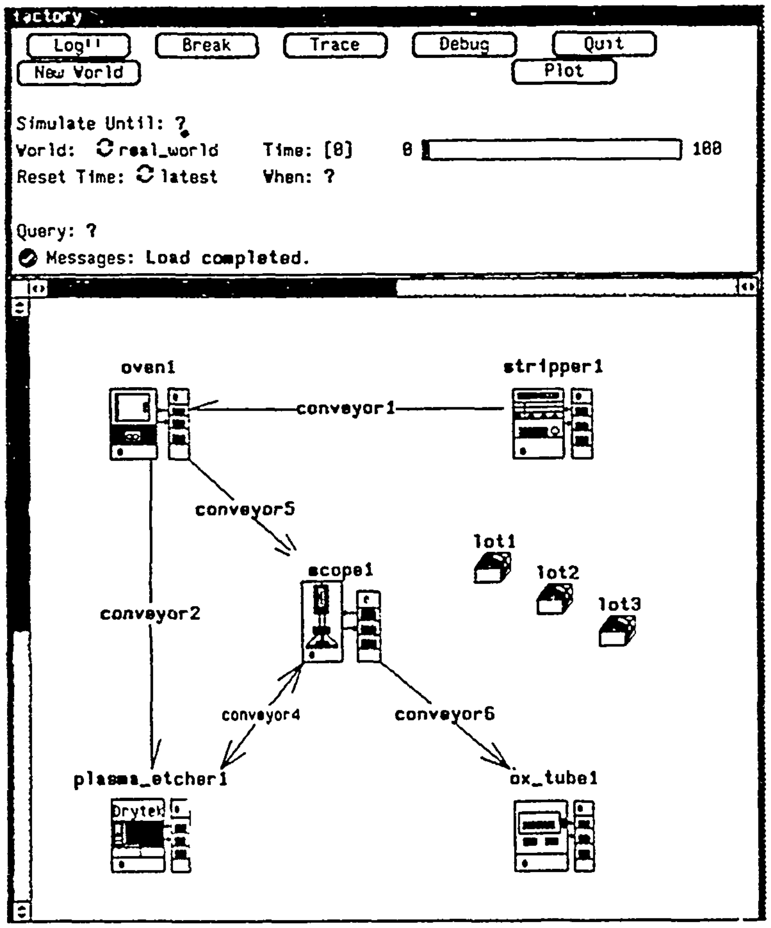
The system serves as a testbed for integration of natural-language and graphics techniques. The philosophy is to let users employ each technology to its best advantage. Graphical interaction, such as that found on the Apple Macintosh, is effective when the objects of interest are on the screen, or are easily found, and when the range of possible actions to be taken is relatively small. Natural language is most appropriate when the user does not know which objects satisfy his needs, or when there are too many to conveniently represent them all iconically. In Shoptalk, the user can employ natural-language descriptions to select a relevant object or set of objects. For example, a user can ask a question like "What is the history of each defective lot that was baked since Wednesday?" without necessarily knowing what defects are being discussed, or which lots were baked since Wednesday. He can then use graphical operations to obtain information once the objects are represented on the screen. This ability to intermix natural-language descriptions with direct manipulation of objects will be a critical component of next-generation user interfaces.

However, a number of problems pose difficulties for each approach. For natural language interaction, these problems include the use of context in resolving anaphoric reference, word sense disambiguation, and the attachment of prepositional phrases, as well as the ability to reason efficiently with quantified statements. Another often-cited weakness of pure natural language systems is the opacity of the system's linguistic and conceptual coverage. Although users know the system cannot understand everything, they do not know precisely what it can understand.

On the other hand, direct manipulation interfaces have difficulties allowing users to apply selected functions to unknown arguments. This weakness is a symptom of their inability to express directly procedures or general information, such as quantificational information, and an inability to describe objects rather than merely to select them or name them. Yet another weakness arises from the necessity of using hierarchies of menus that stand between a user and an action s/he wants to perform. When there are many possible actions, it is cumbersome to have to navigate a menu tree to invoke a known action. The alternative is to type commands directly, or use special function keys, (some systems, such as the Apple Macintosh, provide both of these features in conjunction with direct manipulation). Of course, to use command languages, the user needs to remember the command syntax and functionality; a similar problem exists for remembering the bindings of programmable function keys.

There is clearly a spectrum of possibilities for integrating natural language and direct manipulation. At one end is the NLMenu system at Texas Instruments [23]. Users of that system are able to compose a query by incrementally selecting the set of words from among those that would correctly complete that query given the previously selected words. There is not space here to provide an in-depth discussion of the advantages and disadvantages of NLMenu. However, the lesson we have learned from that system is that we should not necessarily be afraid to let users see what is going on behind the scenes, as long as the system shows only easily understood models of the system's structures. Several intelligent interface systems make use of natural language and direct manipulation components (CUBRICON [18], XTRA [25], Lucy/VSTAT [14, 16], ISI [2]), but they are just beginning to explore the integration of these technologies. As in the systems we will describe, CUBRICON and XTRA allow the insertion of a selected object within a natural language query, as a form of deictic reference, but they do not allow the use of natural language with forms or use the direct manipulation mechanism to constrain context (see section 3).

The techniques described here have been implemented in the Shoptalk system as an extension of Chat [19, 26], a natural language question-answering system implemented in Prolog. Shoptalk is being developed in the domains of semiconductor and printed-circuit board manufacturing. In the latter case, data has been gathered from a factory being developed by the South Carolina Research Authority (SCRA), under contract from the Navy. The SCRA factory is a partially-automated manufacturing environment consisting of automated electronic insertion and test machines, and stations where operators manually insert components and perform various inspection procedures on assembled circuit boards. A typical Shoptalk window is shown in figure 2.1.



```
Simulate Until: ?
World: real_world     Time: [0]     0                    100
Reset Time: latest    When: ?

Query: ?
Messages: Load completed.
```

Figure 2.1: Factory floor window

The system allows users to manipulate manufacturing objects, such as printed-circuit boards or machines on the screen through mouse operations that invoke commands relevant to those objects. Sample commands might be to move lots (groups of semiconductor wafers) from one place to another, or to take a machine offline.

At any time, users can ask English questions about the current state of the factory, and can take actions in which the arguments can either be described or pointed at. The result of the system's analyzing the user's question is a logical form and a set of answers that can be rendered as tables, histograms, and other display forms. The display forms that are presented offer facilities for the user to ask follow-up questions. This facility will be described in detail below.

The remainder of this chapter addresses the Task Requirements, namely to develop a uniform semantics of pointing and a multimodal interface methodology that incorporates natural language and direct manipulation techniques, especially the user of follow-up windows and the direct manipulation of context.

2.3 Uniform Semantics of Pointing

The user should be able to point at objects on the screen and incorporate reference to them into natural language sentences with deictic expressions such as "these" and "this", optionally followed by a head noun (e.g., 'machines'). Shoptalk allows great flexibility in doing so. Users can point virtually anywhere they can type, if it makes semantic sense. First, we present the mechanics of deictic selection and then discuss the semantics.

The user selects objects on the factory floor by clicking the left mouse button within the mouse-sensitive region for that object. Confirmation of selection is made by displaying the object in reverse video. The selection of an object can be cancelled by clicking the left mouse button on it again, and this deselection is confirmed by the object returning to normal video mode.

Multiple objects can be selected simultaneously. Any deictic reference is taken to be to all currently selected objects, and that deictic reference returns those objects to an unselected state: (i.e., the deictic reference cancels or consumes the selection.

The currently selected objects can be used deictically in a sentence by clicking the left mouse button on that sentence. This deictic reference is displayed to the user as a list of the names of the objects, enclosed in square brackets. This list may be edited with the normal editor keys, with the exception that the basic unit is an object in the list, instead of a character. Thus the character-delete key (typically DEL) deletes the last object in the list and the word-delete key (typically CTL-W) deletes the whole list corresponding to a deictic reference. The order in which the selected objects are presented to the user is the order in which the user selected them, with the first object selected appearing first in the list.

When a deictic reference is used in a sentence, a clause is added to the database recording the position of the deictic reference in the sentence (the index of the last character entered) and the list of objects composing that deictic reference. This database clause is used both to simplify updating the display in response to the user's edit commands (above) and to separate out the deictics to be passed to the parser as a separate argument. This argument is an ordered list of the deictic references, with each element of this list being a list of the objects corresponding to that deictic reference. When the deictics are extracted from the sentence, an asterisk is left behind to mark their location.

The grammar includes deictic determiners, and when one is encountered, a corresponding deictic referent of the right type, as constrained by the rest of the sentence, is sought from the list of selected objects. As a result of parsing, the system produces a logical form representation that includes a use of the Prolog expression one_of, which generates successive members of the argument list. For example, in response to the user's typing "Who is operating these <point to [oven1, scope1]> machines?" the system produces the following logical form.

```
answer([C,E]) :-

   exists F G H
     one_of([oven1,scope1],C)
   & {machine(C)}
   & holds(operator(C,E),F,G,H)
   & {person(E)}
   & {strictly_precede(0,G)}
   & {precede(F,0)}
```

Essentially, this expression says that machine C and person E are answers if C is a member of the set of selected machines, and E is operating it. Thus, the semantics of pointing involves the creation and unification of Prolog predicates into a logically based meaning representation. Pointing at objects on the screen introduces *extensional* representations, namely the objects' internal names, into the Prolog expression. Pointing at other locations, such as column headings, produces *intensional*Prolog expressions, as discussed below. In either case, however, the key to handling pointing in this system is to describe the objects being referred to with a Prolog expression.

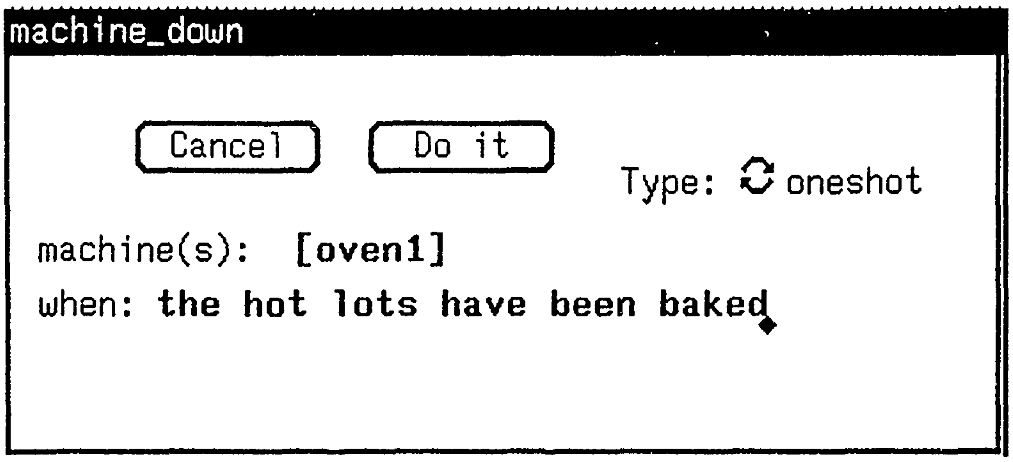
2.4 A Multimodal Interface Methodology

2.4.1 Natural Language Forms

A substantive area of integration between natural language and direct manipulation is *natural language forms,* which allow the user to supply arguments to commands selected from a menu, either in terms of natural language phrases and/or by selecting graphic screen elements. This facility overcomes some of the frequently cited limitations of direct manipulation interfaces: arguments, such as objects or time periods, that are not known or visible can be described, and complex quantificational relations can be expressed more readily in natural language.

For the most part, this facility is used to allow users to issue commands. But, if users were given no guidance about what commands were implemented, they could request actions that the system could not fulfill. However, here we benefit from one of the clear advantages of direct manipulation: the explicit communication of the system's conceptual coverage.

Figure 2.2 shows how one can take down a given machine when a certain condition arises by (1) invoking Machine-down from a menu, (2) pointing at the relevant machine (Oven1) and



Type: oneshot

machine(s): [oven1]

when: the hot lots have been baked

Figure 2.2: Completed machine-down form

depositing the selected machine into the argument What field, and (3) typing "the hot lots have been baked" into the When field. Thus, Oven1 will be taken down when the high priority lots have been baked. The system parses each of these phrases, and assembles a semantic representation for the conditional action. Thus, the system allows the user to fill in slots in an action, describing either objects or time periods. The system parses those expressions separately, and assembles a conditional action. When it can prove that the condition specified in the When field is true, it takes the action.

A second advantage of the use of forms augmented with natural language is the opportunity to specify certain prepositional phrase attachments by filling in particular case roles. Prepositional phrases are syntactically "every way ambiguous" and the number of attachments forms a Catalan series [5][1]. When the case roles marked by various prepositions can be filled in directly, the number of possible parses to be considered can be substantially reduced. For example, consider "Put the block in the box on the table in the corner by the door."

[1]The Catalan series is defined to be

$$Cat_n = \binom{2n}{n} - \binom{2n}{n-1}$$

This sentence has four preposition/noun combinations, and would have 14 parses (= Cat_4). However, for a form with the following argument structure:

Put

> **Object** = "the block in the box"
> **Destination** = "on the table in the corner by the door,"

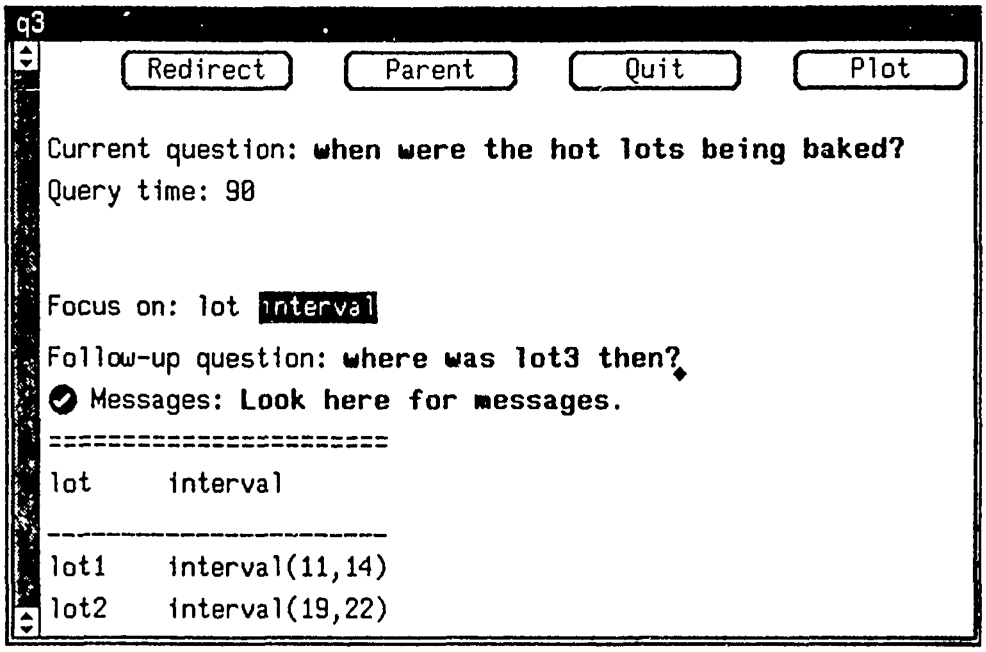
the number of parses can be characterized as $\mathrm{Cat}_1 \times \mathrm{Cat}_3 = 1 \times 5$. With more prepositional phrases the savings may become substantial. One can see this by multiplying the Catalan numbers representing the attachment possibilities that have yet to be made, rather than computing the series for all the prepositional phrases, as would be required if no attachments could be prespecified. In summary, the structure of the form reduces the complexity of the natural language processor's task by making explicit the intended word sense of the action, and by reducing the combinatorics inherent in determining the attachment of the prepositional phrases.

2.4.2 Follow-up Contexts

No one wants to ask just one question. During problem solving, answers to questions lead the questioner to think of still other required information, and this leads him to ask follow-up questions. A characteristic of such questions is the use of anaphora. To date, the determination of the referents for anaphoric noun phrases has been an extremely difficult problem. Shoptalk provides a facility that alleviates some of the difficulty. For the present purposes anaphora will be treated as a unitary phenomenon, although it is well known that pronouns and definite noun phrases behave differently. Many writers have remarked on the need for a pragmatic approach that can draw the needed referent identification inferences from a substantive knowledge base [4, 11, 13]. Others have attempted to constrain that inference process by using focus spaces [8], a focus machine [22], centering [10, 3], and even a purely syntactic approach that is reputed to derive 80-90% correct pronominal referent identifications [12]. In general, the effectiveness of any of the above approaches for resolving anaphoric reference is unknown, and the topic of anaphora in discourse is a locus of substantial research.

We have integrated a number of anaphoric reference techniques by adapting for, use with Chat, Hobbs' method for resolving intrasentential anaphora [12], and by providing a technique for using and manipulating focus spaces via windows. In particular, answers to questions are presented in their own window, thereby graphically limiting context. To ask a follow-up question to a given question, a user must ask the follow-up in the latter's answer window.

To see the *utility* of this technique, consider the following example: The user knows that lot3 is defective, and that the defect was caused by its being baked at too high a temperature. One way to determine which hot lots might have the same defect would be to ask first "When was each hot lot being baked?", to which Shoptalk presents the table in Figure 2.3. The window shows a table whose columns indicate a set of lots that were baked, and the time intervals over which each baking event took place. On the window panel are focus buttons Lot and Interval, indicating that the user can continue to refer to those lots or those intervals if he



```
Current question: when were the hot lots being baked?
Query time: 90


Focus on: lot interval
Follow-up question: where was lot3 then?
Messages: Look here for messages.
======================
lot      interval
----------------------
lot1     interval(11,14)
lot2     interval(19,22)
```

Figure 2.3: Follow-up window

presses the relevant button. For example, acceptable anaphoric expressions referring to the lots include "these," "those," (optionally followed by "lots") "they," "ones" (as in "Which ones ..."), a bare NP (as in "Which lots ..."), or an ellided subject (such as "Which were ...").[2] In the example, to refer to those time intervals, the user might select the Interval button, and type "Where was lot3 then?" Shoptalk interprets "then" as anaphorically referring to the selected time intervals.

Shoptalk constrains the answer to follow-up questions by adding predicates into the questions' logical forms from those parts of the logical form of the prior context that describe the entities that were available for subsequent reference. More specifically, each column heading has a lambda expression describing it. When the user selects the column heading, the system unifies the lambda expression standing behind it into the logical form being built for the question. This approach provides substantial flexibility at the cost of having to retrieve the answers to the previous query again. An extensional approach in which the context contains not descriptions of the answers but the answers themselves works well in many cases, but has the limitation that two sets of entities of the same type in the same answer window cannot be distinguished. The caching approach also has known limitations when one is interfacing to a separate data base; it may bypass the query optimizer by asking for information a tuple at a time. Future work will incorporate both representations of context.

2.4.3 Direct Manipulation of Context

Of course, most natural language database query systems now allow some limited form of anaphoric reference. However, the context mechanisms developed do not provide a full tree structure, as various researchers have argued is needed [9, 21], but rather a bounded linear structure in which users can make anaphoric reference to entities brought into focus by some small number of prior questions and/or answers. A full tree structure is not maintained in part because the semantics of discourse markers (such as "Ok, now"), whose use enables speakers and listeners to navigate the implicit discourse structure tree, is still unclear. We have developed a simple technique allowing the user to avoid these hard problems through the use of the explicit depiction and manipulation of context.

Applying the lesson learned from NLMenu, we decided to display the discourse structure as a tree of queries and to allow the user to view and manipulate the discourse graphically (see Figure 2.4.)

A return to a prior context is effected with a mere selection of a node in the tree. By doing this, two problems from the user's perspective are being addressed. First, the current discourse context is made apparent to the user, allowing the user to decide explicitly whether or not the present question should follow logically the answer to a prior one. Second, by displaying the discourse as a tree, the user can follow up on any query in the discourse, not just those recently asked. From the system's perspective, by allowing the user to see the discourse and select a context within the discourse, the system can determine to what prior question the newly entered question is intended as a follow-up. As a result of this technique,

[2] A subject for future research is the use of definite reference within follow-up windows.

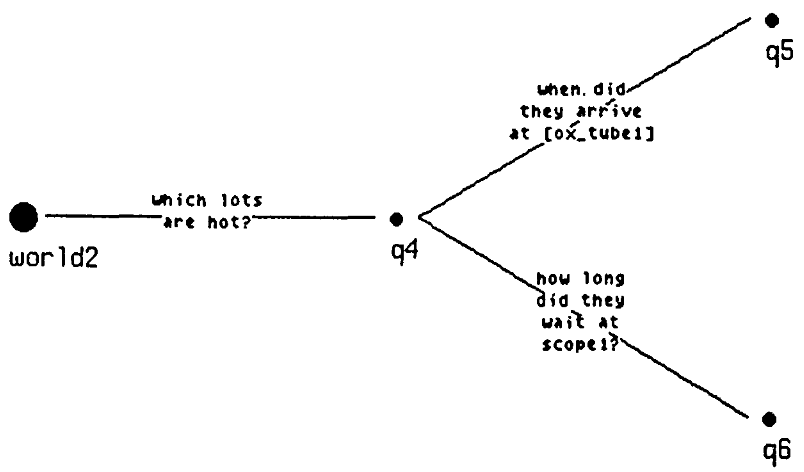

world2
which lots are hot?
q4
when did they arrive at [ox_tube1]
q5
how long did they wait at scope1?
q6

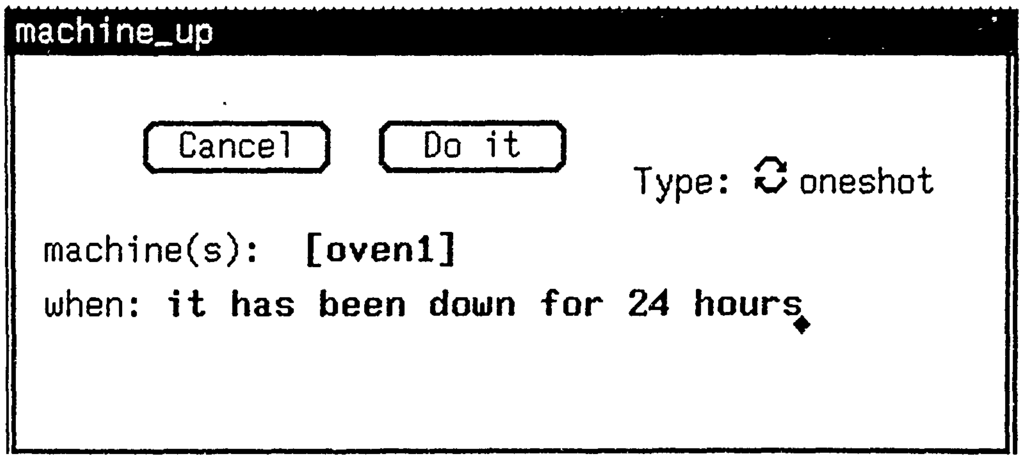
Figure 2.4: A Context Tree

users need not be impeded by the lack of a semantics for cue phrases; they can be served adequately by graphically navigating the discourse tree.

2.5 Example: Scenario Planning

A truism is that the best laid plans will change. Among the many causes of change in military planning are machine breakdown, new orders, changes in priorities, unavailability of personnel and supplies, and, of course, unexpected actions from adversaries. Decision-makers need to cope with this change on an ad hoc basis, but few tools have been developed to assist in this process. What is needed is a tool that allows decision-making personnel (not AI or other systems specialists) to pose What-if scenarios flexibly, and to draw comparisons easily.

Shoptalk was developed to address these issues. Let us compare two hypothetical factory scenarios. After asking a series of questions and determining that a particular machine, say the oven, needs calibration, the user wants to compare two operating strategies for the next day. Assume for the moment that one can operate the machine in a satisfactory state for a short time, but the machine needs calibration for 24 hours to ensure longer-term satisfactory performance. So, in the first scenario, as there is some discretion available on when to take the machine down, the user wants to push as many high priority lots through the oven as



Type: oneshot

machine(s): [oven1]

when: it has been down for 24 hours

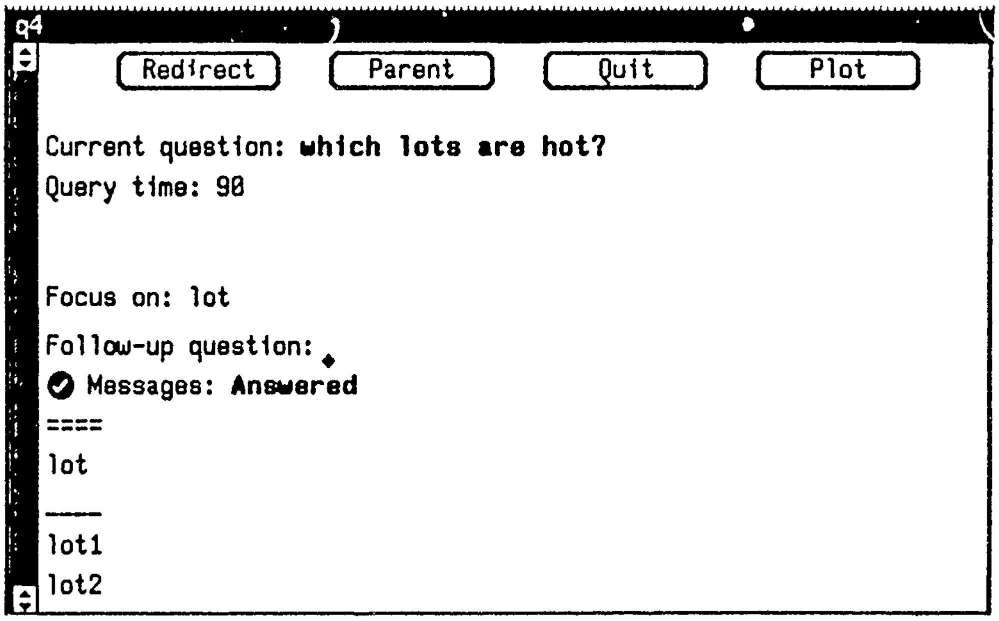
Figure 2.5: Bringing the oven up after it has been down for 24 hours

possible before deactivating it. In the second scenario, the user will just take the oven down immediately and cope with the effects.

Thus, the user has a complex decision to make, but needs to do so relatively rapidly. Here is how it would be done with Shoptalk. First, the user would push the New World button to get a new partition of the database. Then, the user decides when the simulation should stop. In most simulators, the stopping condition is specified as a time period (a day, week, etc.) With Shoptalk, the user can state an arbitrary stopping condition in English. Let us assume the user wants to simulate until five lots have arrived at the oxidation tube. He selects the Simulate button and types " 5 lots are here <point to ox-tube1>" into the Until slot. Then, the user would select the Machine-down action from the action menu, and then fill in the resulting form as in Figure 2.2.

Next, the user needs to return the machine online 24 hours later. Since he does not know when it will go down, he cannot specify an exact time to bring it up. But, with Shoptalk, the user can use the expression, it has been down for 24 hours, as in Figure 2.5.

These actions will take place during the simulation when the specified condition holds. The user does not need to know when that condition will arise, nor precisely which objects satisfy his description. Furthermore, the user has not had to write pattern-action rules in a formal language, nor ask an AI specialist to write special-purpose code to answer the "what-if"



```
Current question: which lots are hot?
Query time: 98


Focus on: lot
Follow-up question:
✓ Messages: Answered
====
lot
----
lot1
lot2
```

Figure 2.6: "Which lots are hot?"

question.

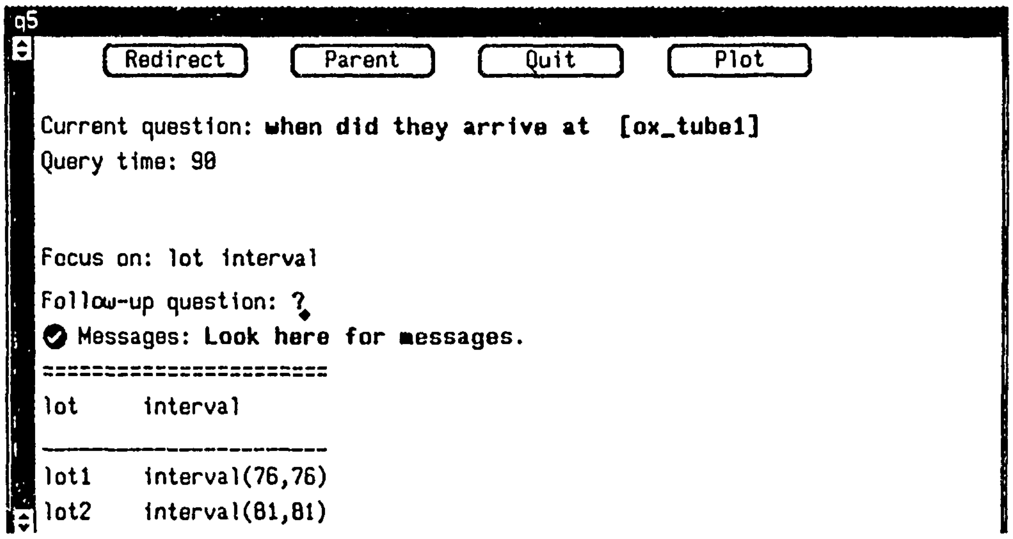
Notice that without the English-language aspect of the interface, the user would not have been able to specify this scenario without having to know when the hot lots would be finished baking, etc. Thus, the interface allows users to be more insulated from details they would rather not know.

Once the factory's future state has been simulated, the user can then use the system's query facilities to elicit information of interest. In our case, the user might first want to identify the hot lots by asking "Which lots are hot?" Figure 2.6 shows the system's response in a follow-up window.

Then, the user follows up on those lots by pushing the Lot button, and asks "When did they arrive here [ox_tube1]?" Figure 2.7 shows Shoptalk's response to this question.

The user sees that the lots in question arrived at time 76 and 81 respectively. The user can compare these results with a similar analysis for the other scenario (taking the oven down now) by creating a new hypothetical world, editing the actions from the previous scenario to take the oven down now, and running the simulation again.

To compare answers to the same question, the user can retype the old question, or drag the old question into the new world. In the latter case, one can re-ask questions that have been determined contextually and are arbitrarily deeply embedded in the question tree of



```
Current question: when did they arrive at  [ox_tube1]
Query time: 90


Focus on: lot interval
Follow-up question: ?
Messages: Look here for messages.
=====================
lot     interval
---------------------
lot1    interval(76,76)
lot2    interval(81,81)
```

Figure 2.7: "When the hot lots arrived at ox_tube1"

some other world. Thus, one need not compare the first question one has asked, but an arbitrarily complex one that one has developed over time.

In addition to being able to view the final state of the simulation, the user can roll back the simulation to a specific time, or to a time that satisfies some condition (such as when a given lot was being baked), to view the state of the factory at that time. This ability is made possible by a declarative graphics subsystem, and is, to our knowledge, unique. It illustrates in one simple comparison the power of natural language and one weakness of direct manipulation. If the system has simulated for a considerable period of time, the size of the pixels themselves may represent two hours of simulated time, far too coarse a grain size for many problems. There are just too many time points to be selecting from. However, by describing the times of interest, e.g., the times when a lot arrived at the oven, the user can describe many times at once.

2.6 Shoptalk Capabilities

Shoptalk provides capabilities designed to address the needs of decision makers. These capabilities result from the application of a next-generation integrated interface with databases and simulators via a common underlying knowledge representation. A listing of salient features follows.

Integrated Interface

- Natural language question-answering with full use of temporal expression
- Access to factory's past, present, and future
- Declarative knowledge-based graphics, so that graphical actions happen when graphical facts are asserted into the database; temporal indexing of these facts allows for redisplay of graphics for specific times
- Full integration of natural language with graphics, e.g., pointing allowed anywhere within a question or action
- Follow-up questions using pointing or pronouns to indicate user focus
- Context tree allowing for user exploration of different lines of inquiry
- Connection to external databases

Simulator

- What-if scenario planning
- Simulator can stop when an arbitrary condition arises

- Multimodal actions, actions, such as taking a machine down, that are invoked via menus, with natural language and/or pointing to specify arguments
- Conditional actions that take effect when a condition, usually specified in English, arises
- Standing Actions that take effect whenever a condition arises
- Multiple worlds provide a tree-structured segmentation of the database to allow for simulation with different assumptions; tree can be navigated
- Comparison facility allows answering of questions, potentially deeply embedded in a context tree across different worlds
- Rewind facility for simulator shows the factory state at any given time, or any time that satisfies a given condition

In summary, Shoptalk provides an interface combining language and graphics integrated with a simulation via a logically based knowledge representation. The results of this integration are powerful What-happened and What-if capabilities. Many other opportunities exist to combine an integrated interface such as Shoptalk's with other tools for command and control.

2.7 Summary

This chapter has described the uniform semantics for pointing and the overall methodology for integrating natural language processing and direct manipulation techniques, especially as they apply to actions selected from a menu and to the direct manipulation of context. An example of posing and comparing scenarios showed how these different techniques could be combined. To illustrate the integrated interface, please view the accompanying videotape.

Chapter 3

Technical Accomplishments: NLE

The Shoptalk system as originally developed used an extended version of the Chat natural language system [26, 19]. Although highly efficient, the system is hard to extend and has some gaps in coverage, notably conjunction, that would limit its usability. Also, and of major importance, it would be very difficult to extend Chat to become the basis of a spoken language system. For these reasons, it was decided that a new language engine for Shoptalk was needed.

The New Language Engine (NLE) supported by this project has as its ancestor the Core Language Engine (CLE) developed by the SRI International Cambridge Research Laboratory. The NLE grammar has the same format as the CLE, and the CLE parser can be used on the NLE grammar, and -versa. In fact, we view the CLE parser as the development parser, and the NLE as the run-time system. Because of this relationship, a description of the NLE by necessity will require an understanding of concepts from the CLE.

Section 3.1 is taken from the *1987 Annual Report to Alvey Directorate and the Natural Language Processing Club* (NATTIE) under Alvey Project No. ALV/PRJ/IKBS/105 [1], which supported the CLE. It is included here to describe the overall shape of a CLE-style parsing system such as the NLE. More details of the format of the grammar and semantic rules can be found in Appendix A.

3.1 Background: CLE

The CLE is a system for translating natural-language (English) sentences into formal representations of their literal meanings. It is intended to be used as a major system component in applications of natural-language processing. There are two important themes that recur throughout the design of the CLE. One is the use of unification as the basic mechanism for passing information in all phases of processing. The grammar is expressed in a formalism that might be called "phrase-structure unification grammar." The rules are based on a context-free framework, but have feature specifications that can be combined by unification. In the rule formalism, features are specified by means of attribute-value pairs, with unifications indicated by shared variables. Since the system is written in Prolog, categories are compiled into term structures with the principal category symbol used as a functor and each feature paired with a particular argument place, and ordinary Prolog unification is used to implement unification of linguistic categories.

The semantic interpretation rules are also expressed in a unification-based formalism similar to that of the grammar. In these rules, unification is used to build representations of the meanings of sentences in a fully compositional way, in most cases without resort to the usual mechanisms of lambda-abstraction and lambda-reduction. This is done by binding semantic features of constituents to parameterized arguments of their semantic representations, and unifying these features with the semantic representations of the constituents that express their actual arguments.

The other important theme is the use of a technique called "packing", after Tomita [24], for creating compact representations of locally ambiguous constituents at all levels of linguistic processing. The basic idea is that when a particular substring can be analysed in multiple ways, but those analyses all enter into analyses of the rest of the sentence in the same way, the local ambiguities need not be "multiplied out" in building higher level structures. For instance, if the subject noun phrase of a sentence has two analyses and the predicate verb phrase has three analyses, six different structures for the entire sentence would be created by a conventional chart parser. Using packing, however, only one sentence-level structure is created so long as the dividing point between the subject and predicate is the same on all analyses and the features assigned to the top-level constituents for the subject and predicate coincide in all cases. This frequently happens when there are modifier attachment ambiguities, such as occur in iterated noun-noun compounds or with complex post-nominal modifiers.

Although packing was suggested to us by Tomita's work, the technique is implicit in standard polynomial-time context-free parsing algorithms such as Earley's algorithm and the CKY algorithm. We have generalised it, however, so that it can be used with categories represented by arbitrary term structures including variables (rather than just atomic symbols) and we have applied it in the semantic phases of processing as well as parsing.

One important consequence of the use of packing is that we have been able to adopt a staged approach to sentence processing in the CLE design. A staged architecture, in which different levels of linguistic analysis are performed in separate phases, clearly has advantages in terms of development and modification of a language processing system. It means, however, that analyses produced by one phase of processing are not constrained by information available at later phases. With a realistically comprehensive grammar, this often results in an unacceptably large number of syntactic analyses being produced, because semantic and pragmatic information is not available to filter out spurious parses. Packing greatly reduces this problem because the compact representation of local ambiguities makes it possible to compute and represent all the syntactic analyses efficiently. Moreover, we then get the additional benefit that semantic interpretation needs to be carried out only for constituents that form part of a complete syntactic analysis of a sentence.

The current version of the CLE includes four processing phases: Lexical lookup, parsing, semantic interpretation, and quantifier scoping.

Syntactic analyses are produced by bottom-up parsing with top-down constraints, using a "left-corner" parsing algorithm. That is, the parser builds syntactic analyses bottom-up but checks every proposed constituent to make sure that it is compatible with the analysis of the preceding part of the sentence before basing any further analyses on it. The parser explores alternatives depth-first by backtracking but uses a well-formed substring table to avoid reconstructing the same analysis of part of the sentence after bactracking. Packing is implemented by the data structures that encode the well-formed substring table.

The semantic interpretation phase operates in a simple bottom-up fashion, producing all semantic interpretations for all the parses found in the previous phase. Once again, packing is used to represent local ambiguities efficiently. This phase also employs a sortal restriction mechanism, using simple category information to cut down on the number of potential ambiguities in interpreting a particular sentence. For example, in talking about Cambridge colleges, names like "Selwyn" can refer either to a college or to the person the college is named after. So, a sentence like "Did Selwyn found Selwyn?" could have four different interpretations, but only one of them—asking whether the man founded the college—is at all sensible. The sortal restriction mechanism we have implemented associates categories with objects, properties, and relations, so that the only interpretations produced are those which combine properties and relations with the types of objects that are appropriate to them.

Quantifier scoping is treated as a separate phase because it is largely independent of other aspects of semantic interpretation and because it tends to multiply out possibilities that could otherwise be treated simultaneously in considering those other aspects of interpretation. This is illustrated by the sentence "Some college was built by every river." The ambiguity of whether the prepositional phrase should be interpreted as being locative or agentive is independent of the relative scopes of "some" and "every" and can be resolved by a single application of sortal restrictions before the scope possibilities are enumerated.

The quantifier scoping algorithm we have implemented produces all scope permutations that satisfy certain structural constraints but avoids producing multiple readings for simple logical equivalences (e.g., permuting multiple existential or multiple universal quantifiers). The scopings are ordered with respect to linguistic preferences involving factors such as left-to-right ordering and quantifier "strength". For instance, the distinction between "each" and "every" is taken into account, so that the preferred interpretation of "Who founded every college?" is a question asking for the nonexistent single founder of all colleges, whereas "Who founded each college?" is interpreted as asking, for each college, who founded that college.

A more precise description of CLE-style grammars, also taken from the *1987 Annual Report to Alvey Directorate and the Natural Language Processing Club* [1], can be found in Appendix A.

3.2 CLETool

The New Language Engine supported by this project consists of a new CLE-style grammar, semantics, parser and semantic rule interpreter, and a logical form conversion routine targeted at the Shoptalk logical form interpreter. In the course of developing these new components, we found it necessary to develop a graphical grammar development environment for CLE-style grammars. Previously, because of the compilation of CLE-style grammars into Prolog terms, debugging such a grammar involved a laborious hand-tracing of Prolog code. The grammar development environment, termed CLETool, as a tool for developing and debugging CLE-style grammars, increased our productivity several fold.

In the remainder of this section, we first describe the CLETool, and then the grammar, semantics, and parser that were developed. Next, we describe the logical form conversion routine that allowed us to integrate this style of grammar into Shoptalk. Finally, we discuss integration with Shoptalk.

The CLETool is a tool to aid the grammar writer in writing and debugging syntactic and semantic rules and lexical items. It is built on top of the SRI-developoed PLView interface to the SunView window system.

Interaction with the interface is through a main CLETool window, the *base frame*, and associated subwindows. The base frame includes (from bottom to top):

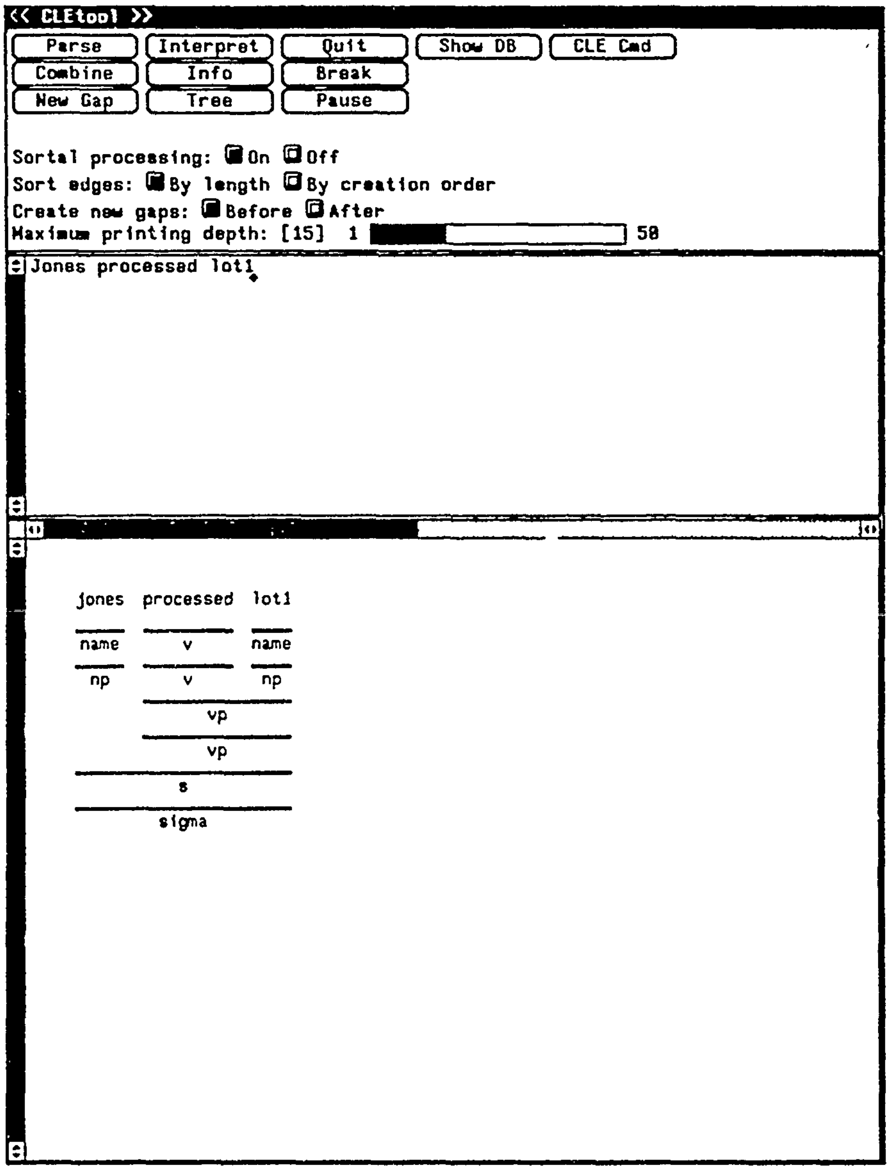
- a canvas for displaying the *chart* associated with the current parsed sentence (the *chart canvas*),
- a text subwindow for listing sentences that are available for parsing, and
- a panel with buttons for operating on and displaying information about selected items in the chart.

Figure 3.1 shows the CLETool base frame, with the chart produced as a result of processing the sentence 'Jones processed lot1'. The chart is displayed as a series of horizontal lines (*edges*) spanning words in the sentence. Each corresponds to a cons' tuent found during the course of parsing. The words in the sentence are displayed above the edges.

Other windows that are created and displayed in the course of using the CLEtool are:

- The *message window*: a small window just below the base frame, used for brief and temporary messages to the user,
- One or several (*auxiliary) information windows* displayed in a staggered fashion to the right of the base frame, and displaying various information about the words or edges in the chart; in particular:
  - *constituent information windows* list textual information about the constituent associated with an edge.
  - *constituent tree windows* display a packed tree associated with the edge.
  - *edge combination windows* display the results of manually combining two edges.

Items on the chart canvas (the lexical items and edges) can be selected by clicking on them, at which point they will have a *select mark* displayed to the left of the textual part of their graphic representation, a small filled circle. Figure 3.2 shows the chart subwindow of the base frame with the 'sigma' edge selected.



Sortal processing: [✓ On] Off
Sort edges: [✓ By length] By creation order
Create new gaps: [✓ Before] After
Maximum printing depth: [15] 1 58

Jones processed lot1

```
jones   processed   lot1
name        v       name
np          v       np
               vp
               vp
            s
          sigma
```

Figure 3.1: The CLETool

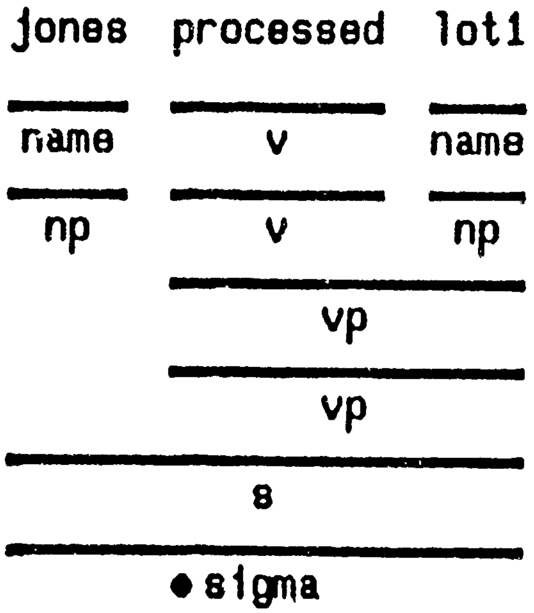


Figure 3.2: Selecting an edge

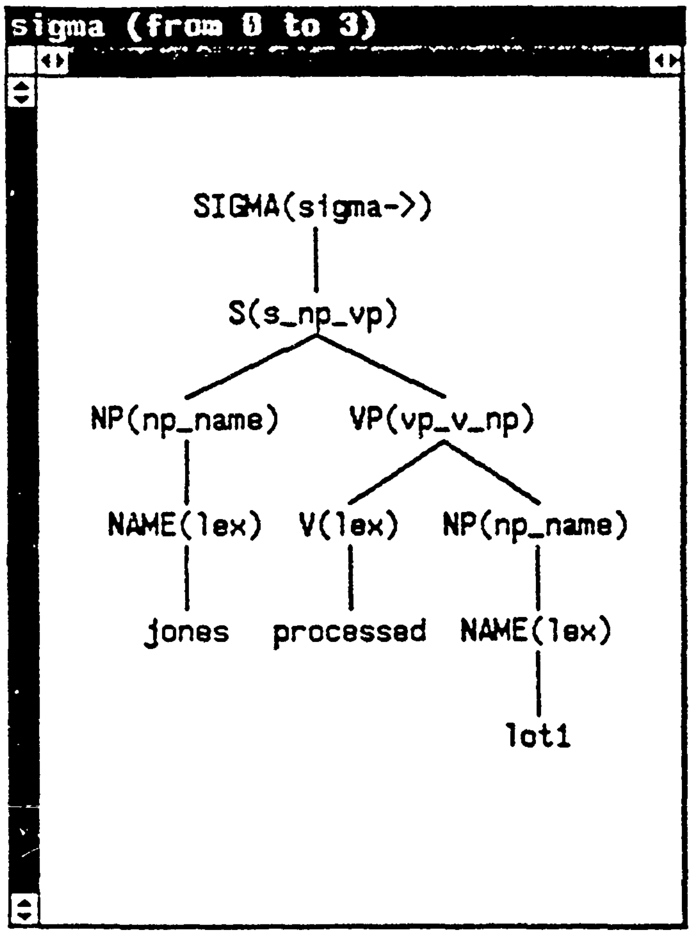
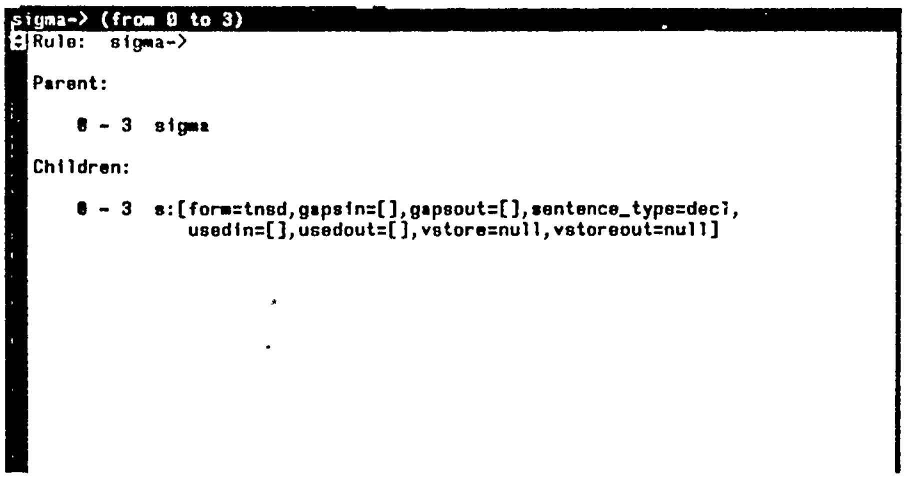


Figure 3.3: Displaying a tree

Various kinds of information about selected edges can be displayed. For example, the packed syntactic tree corresponding to a particular edge can be displayed in a constituent tree window by selecting an edge and then clicking on the button labeled Tree. For the sigma edge, the constituent tree window displayed as a result of clicking on the Tree button is shown in Figure 3.3. Since the sentence is unambiguous, there is only one tree; when the tree for an ambiguous sentence is requested, a packed tree is displayed.

The nodes of the tree are mouse-sensitive; clicking on a node in the tree displays syntactic information about the node and its children and about the syntactic rule that was used in forming the parent constituent. For example, clicking on the sigma node of the tree causes the information in Figure 3.4 to be displayed. The heading Rule contains the rule identifier label for the rule that was used to form the constituent; in this case, the rule identifier label for the rule that was used to form the sigma constituent is "sigma->".

The heading 'Parent' contains information about the syntactic category and syntactic features of the parent category of the rule. Here, the parent spans string positions 0 to 3; its category is 'sigma', and it has no syntactic features.



```
sigma-> (from 0 to 3)
Rule:  sigma->

Parent:

    0 - 3  sigma

Children:

    0 - 3  s:[form=tnsd,gapsin=[],gapsout=[],sentence_type=decl,
              usedin=[],usedout=[],vstore=null,vstoreout=null]
```

Figure 3.4: Tree structure information

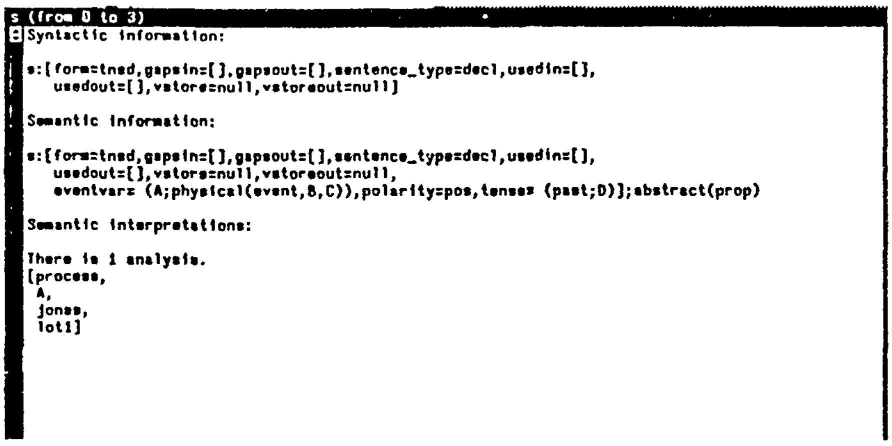
As the rule "sigma->" is a unary phrase structure rule, there is only one daughter; it spans string positions 0 to 3 and is of category "s." The syntactic features associated with the child node are also displayed within square brackets, separated from the category label by a colon.

Besides the syntactic tree associated with an edge, information as to the syntactic and semantic features associated with an edge is displayed in a constituent information window. If the 's' edge in the chart is selected, and the user then clicks the mouse on Info, the constituent information window shown in Figure 3.5 is displayed. The syntactic features associated with the edge are displayed under the heading 'Syntactic Information'; again, the category label is 's', and the features are enclosed in square brackets.

The semantic features are displayed under the heading Semantic Information. The semantic features consist of the syntactic features, some of which may have been further instantiated in the semantic processing phase, together with the features relevant only to semantic processing. Additionally, the sort of the semantic constituent is displayed; in this case, the sort of the sentence "Jones processed lot1" is *abstract(prop)*, the type of a proposition.

The semantic interpretation is also displayed. Since the sentence is neither syntactically nor semantically ambiguous, there is only one interpretation: the predicate "process" applied to three arguments. The first of its arguments is the event argument; the other two are "Jones", the semantic interpretation of the subject of the sentence, and "lot1," the semantic interpretation of the object.

The information displayed by the commands discussed thus far is useful to the grammar



```
s (from 0 to 3)
Syntactic information:

s:[form=tnsd,gapsin=[],gapsout=[],sentence_type=decl,usedin=[],
   usedout=[],vstore=null,vstoreout=null]

Semantic information:

s:[form=tnsd,gapsin=[],gapsout=[],sentence_type=decl,usedin=[],
   usedout=[],vstore=null,vstoreout=null,
   eventvar= (A;physical(event,B,C)),polarity=pos,tense= (past;D)];abstract(prop)

Semantic interpretations:

There is 1 analysis.
[process,
 A,
 jones,
 lot1]
```

Figure 3.5: Constituent information

writer for checking cases where the sentences to be analyzed parse successfully and receive the expected interpretation. Also, the facilities described above are useful in detecting cases in which the grammar overgenerates or where spurious interpretations are produced.

The CLETool is also helpful in debugging the grammar, in cases where a sentence to be analyzed does not receive an expected parse or an expected interpretation. Suppose, for example, that we expect the (ungrammatical) sentence "Jones are processing lot1" to parse. For this unsuccessful parse, the chart canvas is shown in Figure 3.6.

Suppose that we want to determine why this parse was unsuccessful. We can build new edges by selecting edges on the chart and clicking on the command `Combine`. If there is a way to legally combine the edges according to some rule in the grammar, a new edge will appear on the chart canvas. For example, by selecting the rightmost edge and clicking on `Combine`, a new NP edge is formed; this is because there is a grammar rule whose parent is NP, whose child is Name, and whose child category unifies with the rightmost edge on the chart. The result of this combination is seen in Figure 3.7. When a combination is successful, a window such as the one in Figure 3.8 pops up, containing information about the syntactic categories of the new parent edge and the daughter edge or edges. In this case, the rule that engendered the new edge has the rule identifier label "np_name"; the parent category is "np," and the child category is "name." Syntactic features are also displayed.

By repeating this process several times, we find that the verb phrase "are processing lot1" is a legitimate verb phrase; the chart is now as seen in Figure 3.9. However, we find that it is not possible to combine the leftmost NP edge with the newly created VP edge. The CLETool

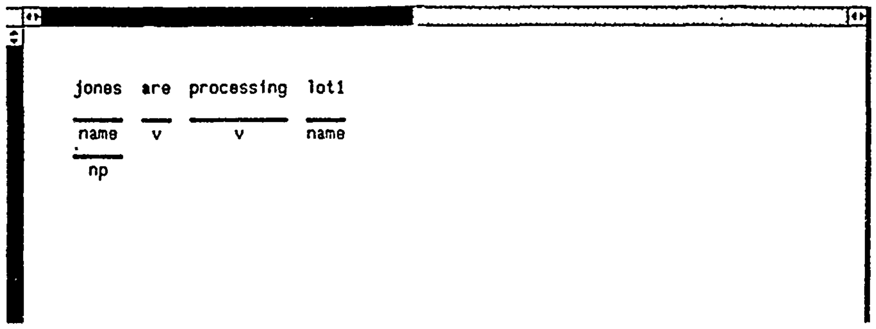


Figure 3.6: An unsuccessful parse

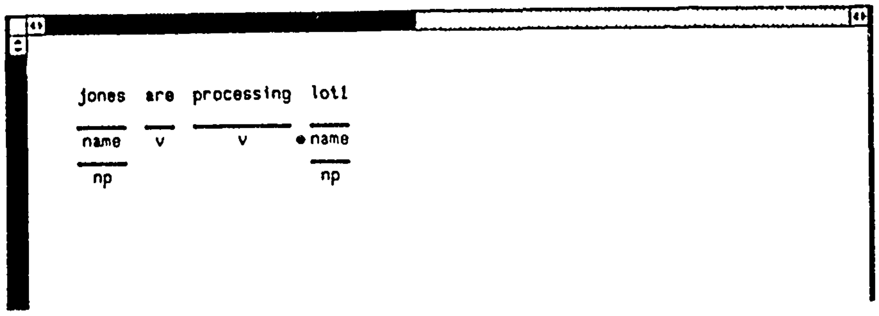
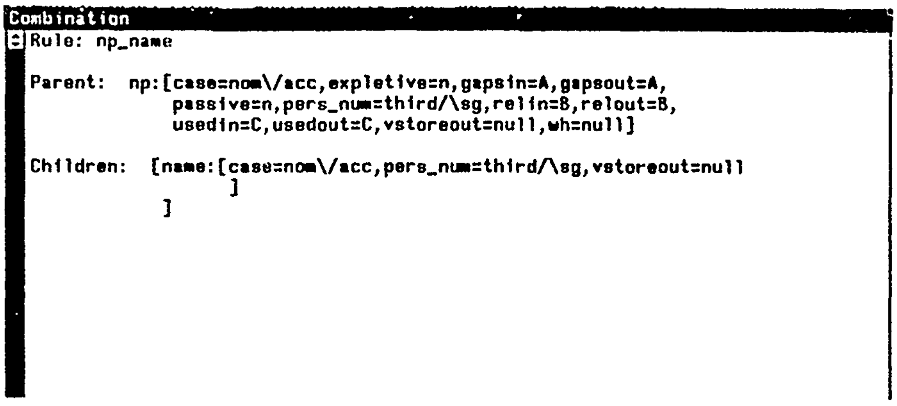


Figure 3.7: Producing new edges with `Combine`



```
Combination
Rule: np_name

Parent:  np:[case=nom\/acc,expletive=n,gapsin=A,gapsout=A,
             passive=n,pers_num=third/\sg,relin=B,relout=B,
             usedin=C,usedout=C,vstoreout=null,wh=null]

Children:  [name:[case=nom\/acc,pers_num=third/\sg,vstoreout=null
                 ]
           ]
```

Figure 3.8: Information about New Edges

can determine why this combination is not allowed.

If the `Combine` command is not successful, it is possible to specify that a specific rule be tried in combining the two edges. Holding down the right button with the mouse cursor on `Combine` displays a chart of the rules that are currently being used; one of these rules can be selected for use by moving the mouse cursor over that rule and then releasing the mouse button. If the rule does not succeed, a window displays the result of attempting to use the rule and an indication of why the rule was unsuccessful.

In the case at hand, selecting the rule whose rule identifier label is "s_np_vp" produces the result in Figure 3.10. The window displays the category from the selected edge, the corresponding category in the rule that was used when the combination was attempted, and an indication of the position at which unification failed. In this case, the feature "pers_num" with a Boolean value failed to unify; the person/number features for "Jones" are incompatible with those required for the subject of "are processing lot1."

Other types of debugging possibilities also exist. For instance, it is possible to propose a gap at any place in the string in order that rules for constituents involving gaps can be tested; this is accomplished by selecting a constituent and clicking on the button marked `New Gap`. The gap will be proposed either before or after the constituent, depending on the setting for the option `Create new gaps` at the bottom of the button panel.

It is also possible to force semantic interpretations for edges; this is accomplished by clicking on the button `Interpret`.

The CLETool provides a powerful environment for grammar writers. Without it, the grammar to be described below would not exist as it would have been too laborious to develop and debug. Future extensions to this environment will involve further support for

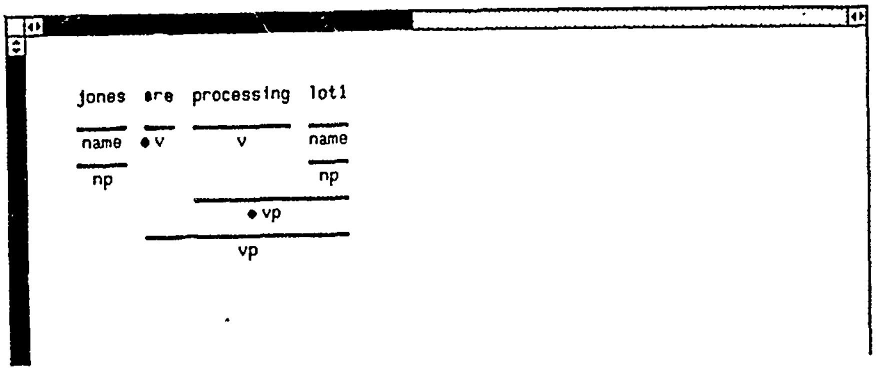
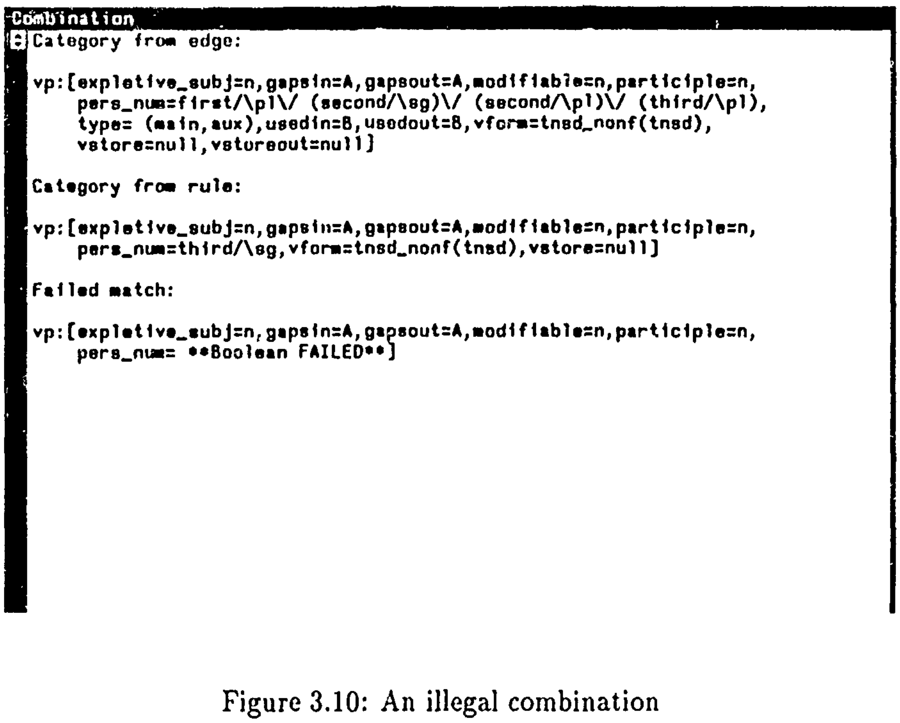


Figure 3.9: A more complete chart


Combination

```
Category from edge:

vp:[expletive_subj=n,gapsin=A,gapsout=A,modifiable=n,participle=n,
    pers_num=first/\pl\/ (second/\sg)\/ (second/\pl)\/ (third/\pl),
    type= (main,aux),usedin=B,usedout=B,vform=tnsd_nonf(tnsd),
    vstore=null,vstoreout=null]

Category from rule:

vp:[expletive_subj=n,gapsin=A,gapsout=A,modifiable=n,participle=n,
    pers_num=third/\sg,vform=tnsd_nonf(tnsd),vstore=null]

Failed match:

vp:[expletive_subj=n,gapsin=A,gapsout=A,modifiable=n,participle=n,
    pers_num= **Boolean FAILED**]
```

Figure 3.10: An illegal combination

the development of semantic rules.

The remainder of this chapter describes various changes and improvements to the CLE made in the course of developing its successor, the NLE.

3.3 NLE Lexicon

The NLE differs from the CLE in that lexical templates are now used in constructing the lexicon. A single lexical template is created for each lexical category; one for each verbal subcategorization frame, for example. Lexical entries make reference to these templates. For example, the following is the lexical template for the transitive verb "add."

```
lex_entry(add, lex(v_trans_base), [sense(v_trans_base_sen, add)]).
```

This template makes reference to base entries for syntactic and semantic templates for transitive verbs, as in:

```
syn_paradigm(v_trans_base,
  v:[type=(main,non_aux), subcat=trans, vform=tnsd_nonf(_),
     pers_num=(\(third/\sg))]).

sen_paradigm(v_trans_base_sen,
      v:[subcat=trans,subjval=S,objval=O,vform=tnsd_nonf(_),
         eventvar=E,tense='LF'(present)],
      [sense_predicate,E,S,O]).
```

An approach of this nature is advantageous because changes to the form of lexical entries can be made to the templates rather than to individual lexical entries. As such, lexical entries are abbreviated, referring only to template names. Creating a lexicon file involves expanding these abbreviated entries with reference to the template entries. This process has been automated, expansion of the lexical entries into their full form is carried out only when either the template files or the abbreviated lexical entry files have been modified.

3.4 NLE Grammar

The NLE grammar is *substantially* larger than the Chat grammar it replaces in Shoptalk, and its coverage is significantly greater.

A number of verb subcategorization frames are handled by the NLE. The Chat grammar allowed only four subcategorization frames. frames for intransitive verbs, for transitive verbs, for ditransitive verbs, and for verbs subcategorizing for predicative complements. The new grammar handles all of these kinds of verbs. Additionally, the new grammar allows for verbs subcategorizing for VP complements; for PPs; for NPs and PPs; for particles, including coverage of both intransitive and transitive verb/particle combinations; for NPs and adjective phrases; and for adverb phrases.

Auxiliary verbs are also treated differently. In the Chat grammar, auxiliaries were treated as a separate syntactic category, with special syntactic rules for combining auxiliaries with the verb phrases they subcategorize for. In the current grammar, auxiliaries are treated with the same rules that are used for other verbs that subcategorize for VP complements, such as 'want' or 'prefer'. For this reason, a smaller set of rules covers a wider range of constructions in the NLE grammar than in the CHAT grammar.

Coverage of coordination is more complete in the CLE grammar than in Chat. The Chat grammar did not allow left-recursive rules such as 'NP $\rightarrow$ NP and NP', where the first symbol in the righthand side of a grammar rule is identical to (or unifiable with) the symbol on the lefthand side of the rule. Such rules had to be written in such a way as to circumvent the left recursion problem.

With the NLE grammar, on the other hand, there is no such problem; left-recursive rules can be written and are handled by the parser. For this reason, rules involving coordination are easy to write, and the coverage of coordination is for this reason much broader. The Chat grammar handled only coordination of relative clauses; the NLE handles coordination of noun phrases, noun, adjective phrases, and prepositional phrases.

VP modifiers are subdivided into groups in the NLE grammar, permitting distinctions between kinds of VP modifiers that were not made in the Chat grammar. For instance, VP modifiers cannot in general appear between an auxiliary verb and a main verb. We can ask, 'Which lots were baked before noon?' but not, *'Which lots were before noon baked?' However, certain adverbs are allowed to occur in those positions; for instance, 'Which lots will next go to the stripper?' The difference between these kinds of adverbial modifiers is represented in the NLE grammar in terms of syntactic features.

Dates and times were not covered in the Chat grammar. NLE rules have been written for parsing various date and time constructions, e.g., 'twelve noon,' 'six pm,' and 'January seventeenth'.

The semantic coverage of the NLE grammar is also broader than that of the Chat grammar. NLE semantic rules corresponding to the syntactic rules described above have been written, so the constructions described above as not being covered by Chat are given complete coverage, both syntactic and semantic, by the NLE grammar.

The treatment of tense and aspect in the NLE grammar is also different. Unlike the treatment in Chat, the treatment is scopal: tense and aspects are treated as operators that scope over part or all of the logical form for a sentence. Some non-tense-bearing elements are interpreted temporally; for example, a 'hot lot' is a lot that is hot at a particular time. Such elements can be interpreted either with respect to the tense and aspect of the sentence in which they appear, or with respect to some pragmatically relevant time such as the present. Whether or not these elements are interpreted with respect to the tense and aspect of the tensed verb of the sentence is determined by whether they appear inside or outside the scope of the tense/aspect operators.

3.5 Logical Form Conversion

Because NLE is a general natural-language processing tool that is intended to be transportable among domains and applications, it has been designed to produce logical forms motivated entirely by linguistic considerations and not by the demands of any particular application. As a consequence, the logical forms produced by NLE look quite different from those produced by the CHAT system employed by the initial version of Shoptalk. A system had to be implemented to produce a translation from one formalism to the other.

3.5.1 Characteristics of NLE logical form

NLE logical forms represent tense as scoped operators that range over parts of logical form governed by the time reference indicated by the tense operator. For example, a query such as "When were the hot lots being baked?" would be represented in NLE by the following logical form.

```
[whq,
  quant(exists, A, [event, A],
         [past, [prog, quant(wh, B, [time, B],
                                    quant(the, C, [and [lot, C], [hot, C]],
                                  [and, quant(exits, D, [agent, D],
              [bake, A, D, C]),
    [interval, A, B]]))]])]
```

In the above logical form the tense as introduced by the auxiliary and verb sequence "were being baked" is represented as the operators "past" and "prog" being applied to the remainder of the logical form. Predicates that denote events have associated event variables, of which intervals can be predicated during which they take place. The answer to this query is a description of such an interval associated with the baking event for a set of hot lots. Some predicates representing states (such as "hot"), for linguistic reasons do not have associated event variables; however, they nevertheless can be evaluated with respect to different points in time. The time points with respect to which the various temporally evaluable predicates are evaluated are determined by their scoping with respect to the temporal operators.

The Chat version of Shoptalk represented all tense information in a manner similar to that introduced by Reichenbach [20]. (See [6] of Shoptalk's time and tense analysis.) Each predicate is assumed to hold during an interval with starting and ending points. Tense operators introduce various constraints on the endpoints of the intervals. For example, a p. progressive sentence like the one in this example indicates that events such as "bake" in the scope of the tense operator must be constrained so that some point in the interval during which the hot lots were baked precedes the current simulated time. (The baking could, but does not have to, continue into the future).

There are many other differences between Chat and NLE logical forms. In NLE superlatives are represented by a "greatest-degree" operator applied to a predicate in the context of a restrictor. For example, "the hottest lot," represented as

```
[greatest_degree, [hot, X], [lot, X]],
```

must be translated into an aggregating operation that collects all the lots and their associated priority, and a maximizing operation that returns a maximal subset.

3.5.2 An Overview of the Translation Process

The translation of a logical form fiom NLE to a Prolog clause form that can be evaluated takes place in two passes.ffi

- First Pass

  1. Translate quantifier expressions, and determine whether variables are to be included in the answer list.
  2. Introduce enumeration operators for numeric quantifiers such as "how many."
  3. Translate superlatives into appropriate aggregating and maximization operations.
  4. Introduce temporal constraints for temporally evaluable predicates. Introduce quantified variables for interval endpoints.
  5. Translate each temporally evaluable predicate as a "holds" predicate over the interval determined by the temporal operators in whose scope the predicate lies.
  6. Eliminate sortal predicates introduced by NLE but not used by the Shoptalk simulator, such as "event" and "agent."

- Second Pass

  1. Translate temporal constraint predicates such as "before" and "after" from constraints on the event's associated event variable to constraints on the event's interval endpoints.
  2. Eliminate NLE event variables.
  3. Translate the NLE application-independent predicates into the specific predicates used in the Shoptalk simulator.

After both passes of the logical form translation have been completed, the resulting logical form is logically equivalent to what would have been produced by Chat for the same input sentence. Because NLE has more comprehensive coverage than Chat, it is possible to process some sentences with NLE that were not processed by the Chat-based Shoptalk system, such as, for example, imperative commands.

3.6 NLE Parser

As part of the NLE effort, a new parser was designed and implemented. The new parser is an adaptation, for parsing keyboard-entered text, of a parser originally designed for use in speech understanding systems [17]). There are two main advantages of the NLE parser over the CLE parser:

- It is compatible with the speech understanding system being developed at SRI, which should facilitate the eventual development of a spoken-input version of Shoptalk
- Several features introduced for use in speech understanding make the NLE parser particularly suited to online parsing, that is, parsing while the user is typing

Both the NLE parser and the CLE parser find all syntactic parses of the input, before engaging in semantic processing. This makes the system modular and eliminates some potential redundancies in semantic processing, as only the most general analyses for a particular syntactic category over a particular segment of input are maintained for further processing. The most important difference between the NLE and the CLE parsers is that the NLE parser uses breadth-first search while the CLE parser uses depth-first search. This difference in search pattern has important consequences for online parsing.

The point of online parsing is to reduce the apparent processing time to the user. Suppose a parser takes ten seconds on average to process a ten word sentence. If parsing does not begin until the input is completely entered, then the user will have to wait at least ten seconds before he sees any response from the system, which will make the system seem very sluggish. If the parser is operating as each word is entered, however, the waiting time will only be that needed to process the last word, a second or so, as long as the user's typing speed is less than sixty words per minute. The system will seem far more responsive, even though it is processing words at the same rate.

The depth-first search used by the CLE parser makes it difficult to perform online parsing effectively. Only one parse is followed until the end of the input is reached, at which point the rest of the parsing space is exhaustively searched for other analyses. Thus most of the computation is delayed until all words are present, which minimizes the benefits of online parsing. The breadth-first search used by the NLE parser does as much computation as possible as each word is entered, thus maximizing the effect of online parsing.

Another feature of the NLE parser addresses an important practical issue in online parsing, the need to allow for editing of the input as the user types. Standard parsing algorithms are designed to handle only a single sequence of words. If the user backs up and deletes words that have already been passed to the parser, then some mechanism must be provided to put the parser back into a previous state. The NLE parser, however, was designed to work with a speech recognizer that produces many different hypotheses for what word might appear at a particular position in the input. It does this by building a tree of alternative word sequences, branching from left to right, keeping the analyses of different branches of this tree separate. Thus to accommodate editing of the input it is not necessary to undo any of the work done along the branch that has been edited out. It is only necessary to sprout another branch from the point in the input where type-ahead resumes.

Currently the NLE parser takes about 1.5 times as long as the CLE parser to parse a given input, due to the overhead introduced by the breadth-first search mechanism. This speed disadvantage should be completely swamped by the effect of online parsing, however, and we are also planning further improvements to the parsing algorithm which should make the NLE parser comparable to, or faster than, the CLE parser.

3.7 NLE Semantic Interpreter

The parsing records which the NLE parser creates are quite different from those created by the CLE parser. Since the semantic interpreter uses the parsing records as input, a new NLE interpreter was written to use the NLE-style records during the semantic interpretation phase.

The NLE parsing records consist of information as to constituents which span positions in the input string. The CLE parsing records encode that information; additionally, the CLE records containe information about local tree structure. Specifically, for each rule used during a parse, the CLE parser produces a record consisting of the local tree using that rule, including information about the syntactic features of the mother and the daughters in the rule. Since this information is not recorded by the NLE parser, it must be reconstructed when semantic interpretation is performed.

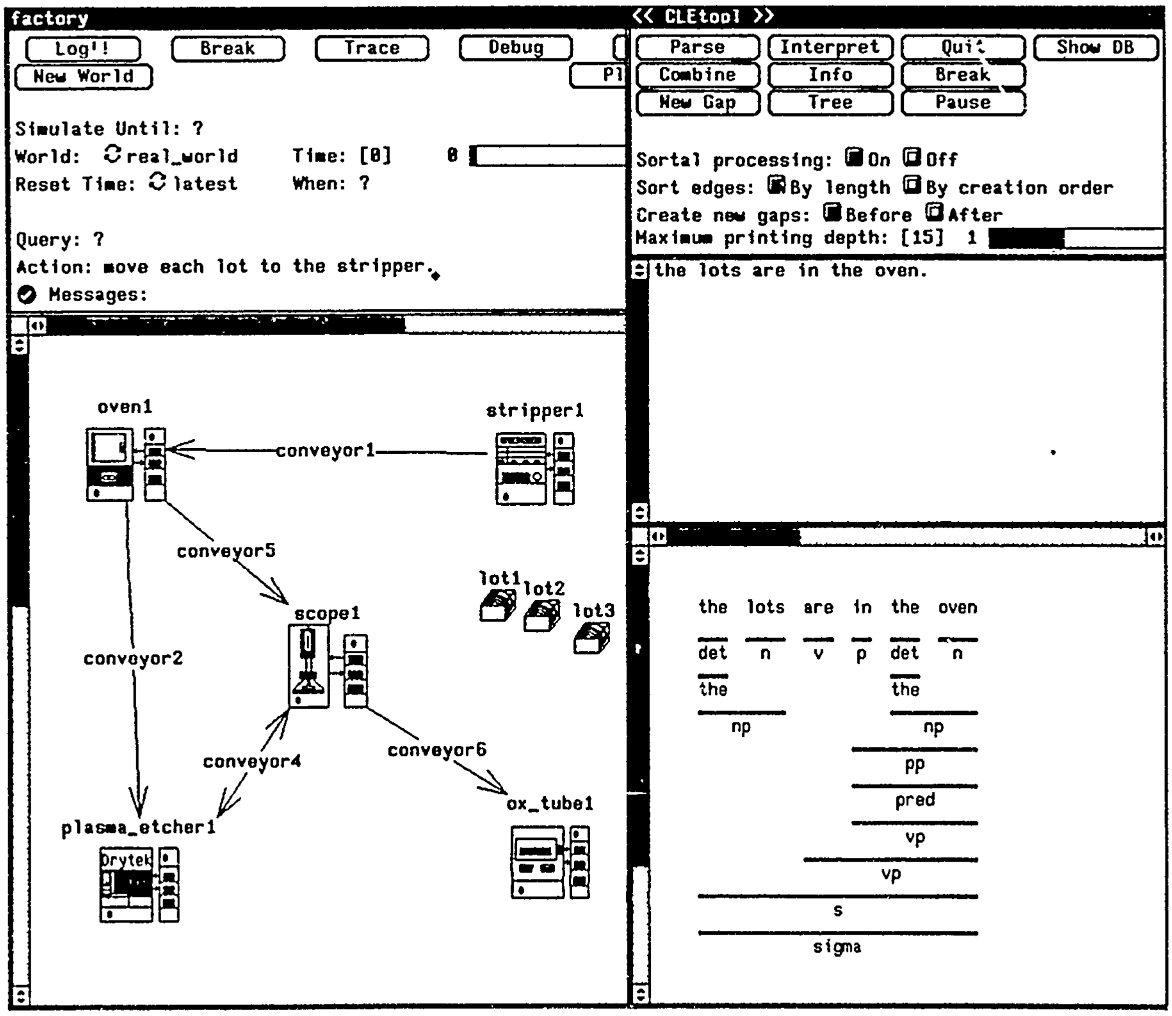
The NLE interpreter differs from the CLE interpreter, then, mainly in that reconstruction of local syntactic trees must be done prior to performing semantic analysis on each local tree of a parse. After this is done, semantic analysis proceeds in a manner very similar to that of the CLE interpreter; semantic rules of the same format are used in both interpreters, and identical semantic record structures are produced as input to the quantifier scoping routine.

3.8 Integration with Shoptalk

Various changes to Shoptalk were made during the course of this project. In order to make viewable videotapes, the system had to be modified to allow multiple families of font sizes, and to allow the fonts in any display to be enlarged. In general, the smallest viewable font is 16 points.

A second change made was to develop an "action editor." When the user has created and run a scenario, and wishes to examine a slightly different but related scenario, the action editor can be invoked. It displays the various standing orders that have been given in one scenario, and allows the user to select or deselect various standing orders to be installed into the new scenario. This speeds up considerably the running of scenario experiments, and reduces error.

A preliminary integrated system, Shoptalk-II, was developed using the NLE grammar, CLE parser, CLETool, and logical form conversion routines (see Figure 3.11). This system has a much wider linguistic coverage than the Chat-based Shoptalk However, currently the



Simulate Until: ?
World: real_world Time: [0] 0
Reset Time: latest When: ?

Query: ?
Action: move each lot to the stripper.
Messages:

Sortal processing: [✓ On] Off
Sort edges: [✓ By length] By creation order
Create new gaps: [✓ Before] After
Maximum printing depth: [15] 1

the lots are in the oven.

Figure 3.11: Shoptalk-II with CLETool

system can only use the CLE parser. The NLE parser awaits various small efforts before it can be installed as planned.

- The use of forms requires that the NLE be asked to parse constituents rather than entire sentences. This has been designed, but remains to be implemented.
- The use of the focus buttons in follow-up windows requires that the semantic interpretation rules of the NLE be able to unify a lambda expression into the logical form currently being constructed. Whether do to this at the level of the converted (Chat-like) logical forms or with the NLE forms directly is an open question.
- The facility to perform parsing-while-typing has been designed but not implemented. The only issue to be worked out is how to handle erasing. Various designs have been proposed, but one has not yet been settled on.
- It is expected that the NLE parser would be faster if it were to use the new Quintus Prolog term subsumption code rather than the user-written code. This conversion remains to be done, but again, should be relatively simple.
- Follow-up windows will be used for intersentential anaphora. Research on intrasentential pronoun reference is under way and will lead to future algorithms [7].

3.9 Conclusion

The Shoptalk system features an advanced user interface that integrates natural language processing and direct manipulation techniques in novel and powerful ways. The system uses the strengths of each technology to overcome the weaknesses of the other. The innovative features of this interface appear to provide significant leverage for the underlying software systems, resulting in new ways of using them (e.g., compare Shoptalk's simulator to other discrete-event simulators commercially available), and appear to support creative problem-solving better than either technology used in isolation. These potential advantages, however, remain to be evaluated empirically. This is one immediate next direction for research that SRI is undertaking on internal research and development in its Computer Dialogue Laboratory. A second direction is to generalize the system to allow for rapid porting to new command and control domains. Finally, a third important direction is the integration and revision of the present Shoptalk interface technologies with spoken language understanding. Because of RADC support of the present effort, this integration is now feasible.

Bibliography

[1] H. Alshawi, R. C. Moore, D. B. Moran, and S. Pulman. Research programme in natural-language processing. Technical Report Annual Report, SRI Cambridge Computer Science Research Centre, Cambridge, U. K., July 1987.

[2] Y. Arens, L. Miller, and N. Sondheimer. Presentation planning using an integrated knowledge base. In *Architectures for Intelligent Interfaces: Elements and Prototypes*, Monterey, CA, March 1988.

[3] S. Brennan, M. Friedman, and C. Pollard. A centering approach to pronouns. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, June 1987.

[4] E. Charniak. Jack and Janet in search of a theory of knowledge. In *Advance Papers of the Third Meeting of the International Joint Conference on Artificial Intelligence*, Los Altos, California, August 1973. William Kaufmann Inc. Reprinted in *Readings in Natural Language Processing*, Grosz, B. J , Sparck Jones, K., and Webber, B. L. eds., Morgan Kaufmann Publishers, Inc., Los Altos, California, 1986.

[5] K. Church and R. Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, 8(3-4):139–149, 1982.

[6] M. Dalrymple. The interpretation of tense and aspect in English. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, New York, June 1988.

[7] M. Dalrymple. *Syntactic constraints on anaphoric binding*. Ph.D. Thesis, Department of Linguistics, Stanford University, in preparation.

[8] B. J. Grosz. The representation and use of focus in dialogue understanding. Technical Report 151, Artificial Intelligence Center, SRI International, Menlo Park, California, July 1977.

[9] B. J. Grosz. Focusing and description in natural language dialogues. In A. K. Joshi, B. Webber, and I. Sag, editors, *Elements of Discourse Understanding*. Cambridge University Press, 1981.

[10] B. J. Grosz, A. K. Joshi, and S. Weinstein. Providing a unified account of definite noun phrases in discourse. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 44–50, Cambridge, Mass, 1983.

[11] J. Hobbs. Coherence and coreference. *Cognitive Science*, 3(1):67–90, 1979.

[12] J. R. Hobbs. Resolving pronoun reference. *Lingua*, 44, 1978. Reprinted in *Readings in Natural Language Processing*, Grosz, B. J., Sparck Jones, K., and Webber, B. L. eds., Morgan Kaufman Publishers, Inc., Los Altos, California, 1986.

[13] J. R. Hobbs and P. Martin. Local pragmatics. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Altos, California, August 1987. Morgan Kaufman Publishers. Inc.

[14] J. Hollan, J. Miller, E. Rich, and W. Wilner. Knowledge bases and tools for building integrated multimedia intelligent interfaces. In *Architectures for Intelligent Interfaces: Elements and Prototypes*, Monterey, CA, March 1988.

[15] L. Karttunen. D-patr: A development environment for unification-based grammars. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 74–80, Bonn, West Germany, 1986.

[16] J. McKendree and J. Zaback. Planning for advising. In *Proceedings of CHI'88*, Washington, D.C., May 1988.

[17] H. Murveit and R. Moore. Integrating natural language constraints into hmm-based speech recognition. In *Proceedings of ICASSP-90*, 1990.

[18] J. G. Neal and S. C. Shapiro. Intelligent multi-media interface technology. In *Architectures for Intelligent Interfaces: Elements and Prototypes*, Monterey, CA, March 1988.

[19] F. C. N. Pereira. *Logic for Natural Language Analysis.* PhD thesis, University of Edinburgh, 1983.

[20] H. Reichenbach. *Elements of Symbolic Logic.* MacMillan, New York, NY, 1947.

[21] R. Reichman. *Plain-speaking: A theory and grammar of spontaneous discourse.* PhD thesis, Department of Computer Science, Harvard University, Cambridge, Massachusetts, 1981.

[22] C. L. Sidner. Towards a computational theory of definite anaphora comprehension in english discourse. Technical Report 537, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, June 1979.

[23] H. R. Tennant, K. M. Ross, R. M. Saenz, C. W. Thompson, and J. R. Miller. Menu-based natural language understanding. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 151 - 158, Cambridge, Massachusetts, June 1983.

[24] M. Tomita. An efficient context-free parsing algorithm for natural language. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 756–764, Los Angeles, California, 1985.

[25] Wolfgang Wahlster. User and discourse models for multimodal communication. In J. W. Sullivan and S. W. Tyler, editors, *Architectures for Intelligent Interfaces: Elements and Prototypes.* Addison-Wesley, Palo Alto, CA, 1989.

[26] D. Warren and F. Pereira. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8(3):110–123, 1982.

Appendix A

Format of CLE Grammars

This section is taken from the 1987 Annual Report to Alvey Directorate and the Natural Language Processing Club (NATTIE) under Alvey Project No. ALV/PRJ/IKBS/105 [1].

A.1 Category and Feature System

A.1.1 Categories for Linguistic Analysis

In the CLE, information about the syntactic and semantic properties of linguistic constituents is represented using complex categories that include a principal category symbol and specifications of constraints on the values of syntactic and semantic features. Categories appear in syntax rules, semantic interpretation rules, and lexical entries. Matching and merging of the information encoded by categories is carried out by unification.

A.1.2 CLE Categories

A category consists of a category symbol and a set of feature-value pairs (or feature specifications) represented as a list:

$\langle$*category-symbol*$\rangle$: [$\langle$*pair*$_1\rangle$,..., $\langle$*pair*$_n\rangle$]

The category symbol is a constant (i.e. a Prolog atom), and the list of feature values may be empty (i.e."[]"). It is not necessary to give an explicit pair for each feature associated with a category symbol; those that do not appear on the list take a default value. The category symbol can be regarded as the value of a distinguished feature giving a coarse classification of constituents. Its special treatment is motivated by implementation and efficiency considerations.

The feature-value pairs in a category may appear in any order. Each pair consists of an atomic feature name and a value, which can be an arbitrary Prolog term:

$\langle$*feature-name*$\rangle$=$\langle$*value*$\rangle$

In particular, feature values may contain variables or categories. For example, in the category represented below, the value of the gapsin feature is a Prolog term consisting of a list structure containing a category with symbol np:

```
s:[type=Type,form=tnsd,gapsin=[np:[num=N]]]
```

Variables appearing in feature values (e.g. Type and N above) are used to express unification constraints on feature values between categories and within the same category, or to stop a feature value from being set to the default specified for that feature. The variables are in fact ordinary Prolog variables, so they start with a capital letter or the underscore character ("_"), and their scope is the largest term in which they occur (e.g., a syntax rule).

A.1.3 Category Unification and Subsumption

Category unification in the CLE is Prolog term unification of the internal representations of the categories (Section A.1.4). This means that the categories must have the same category symbol and that, for each feature associated with the symbol, the corresponding values must unify as Prolog terms, with consistent variable bindings across all the features. The feature values involved take into account any feature default declarations, which are applied when the category is first internalized. Categories appearing as feature values are unified recursively using the same procedure.

For example, in the absence of default declarations the two categories

```
s:[form=tnsd,gapsin=[np:[num=plur]]]
s:[gapsin=[np:[]],form=F,type=ynquestion]
```

will unify resulting in a category equivalent to:

```
s:[form=tnsd,gapsin=[np:[num=plur]],type=ynquestion]
```

If, on the other hand, the feature type had been declared to have a default value declarative, then the unification would fail.

For one category to subsume another they must be able to unify, and each feature value of the subsumed category must be an instance of the the value of the subsuming category, taking into account any feature default declarations. In the above example neither category subsumes the other. Subsumption is a frequent operation in our generalization of "packing" to constituents with complex categories.

A.1.4 Internal Format of Categories

In the internal representation of a category, the category symbol becomes the functor of a Prolog term with two arguments. The first argument is a term with functor syn and arguments giving the value of each syntactic feature associated with the category symbol. Similarly, the second term has the functor sem and the values of semantic features as arguments:

⟨*category_symbol*⟩(syn(⟨*syn-val*⟩,...),sem(⟨*sem-val*⟩,...))

For example the category

```
s:[form=tnsd,qvar=X,gapsin=[np:[num=plur]]]
```

might be translated into:

```
s(syn(tnsd,[np(syn(plur,...),sem(...))],...),sem(X,...))
```

Note that categories appearing as part of feature values are also translated into the internal format.

For any particular category symbol, the value of a syntactic or semantic feature occupies a fixed argument position in a syn or sem term. It is this property of the internal format which allows the unification and subsumption operations on categories to be carried out efficiently as Prolog term unification and subsumption.

A.2 Syntactic Information

A.2.1 Syntax Rules

The basic form of CLE syntax rules is that of phrase structure rules in which categories have feature specifications of the kind described in the previous section. Each rule has the format:

syn(⟨*rule-identifier*⟩, [⟨*mother*⟩,⟨$item_1$⟩,...,⟨$item_n$⟩]).

In most cases, this is just stating that a constituent of category ⟨*mother*⟩ dominates daughter constituents with categories ⟨$item_1$⟩,...,⟨$item_n$⟩ in the order in which they appear in the rule (the exception to this is that an item can be a literal word or a test; see below).

Allowing arbitrary structures with variables as the feature values in categories means that the rule formalism is not limited to describing context free grammars. In practical terms, however, the more powerful formalism is used mainly in order to produce a more compact and perspicuous grammar.

The following example syntax rule has a sentence (s) composed of a noun phrase (np) and a verb phrase (vp). The identifier chosen for this rule is the mnemonic s_np_vp:

```
syn(s_np_vp,
    [s:[type=decl,form=tnsd,gapsin=Gi,
        gapsout=Go,usedin=Ui,usedout=Uo],
     np:[num=N,thirdps=Th,firstp=F],
     vp:[num=N,thirdps=Th,firstp=F,
         form=tnsd,modifiable=M,gapsin=Gi,
         gapsout=Go,usedin=Ui,usedout=Uo]]).
```

Variables are used to express constraints between the daughters (e.g. the variable N enforces number agreement between the noun phrase and verb phrase daughters in this rule) and to pass information up and down a syntactic structure (e.g., by unifying the values of the features gapsin and gapsout that appear on the s and the vp in this rule).

For categories that rewrite as the empty string (i.e., "gap" categories) the rule is simply written without daughter categories:

```
syn(empty_np,
    [np:[num=N,gapsin=[np:[num=N]|Rest],
         gapsout=Rest,usedin=Used,usedout=[g|Used]]]).
```

A.2.2 Syntactic Analyses

Syntactic analyses generated by the CLE parser take the form of phrase structure trees in which mother constituents dominate their daughter subconstituents. Each constituent is labeled with a category. For example, the following syntax tree is generated for the sentence *A bishop founded Selwyn*:

```
[sigma,
 [sigma->,
  [[s(syn(tnsd,[],[],decl,[],[]),A),
    [s_np_vp,
     [[np(syn(n,[],[],n,sing,n,y,[],[],n),B),
       [np_det_nbar,
        [[det(syn(n,sing,y,n),sem(C,D,E)),[lex,a]],
         [nbar(syn(n,sing,y),F),
          [nbar_n,[[n(syn(...),sem),[lex,bishop]]]]]]]]],
      [vp(syn(n,tnsd,[],[],y,sing,n,y,[],[]),G),
       [vp_v_np,
        [[v(syn(n,tnsd,H,sing,trans,y),sem(I,J,K,L)),
          [lex,founded]],
         [np(syn(n,[],[],M,sing,n,y,[],[],n),N),
          [np_name,[[name(syn(...),sem),
                     [lex,selwyn]]]]]]]]]]]]]]
```

This tree is made up recursively out of subtrees of the form:

[⟨*mother-category*⟩
 [⟨*rule-identifier*⟩
  [⟨*daughter-subtree*$_1$⟩, ..., ⟨*daughter-subtree*$_n$⟩]]]

Categories are shown in their internal format (Section A.1.4), where non-lexical categories have variables (which begin with a capital letter) standing for their semantic features. The rule-identifier is one of the following:

1. The identifier of the syntax rule that generated the analysis of the subtree (e.g., s_np_vp or np_det_nbar above).

2. The symbol sigma->, which is the identifier for the notional rules deriving the start categories for the grammar from the distinguished symbol sigma.

3. The symbol lex, indicating a terminal node allowed by a lexical category, in which case the lexical item replaces the list of daughter subtrees.

The empty list [] can also appear as a terminal symbol in the tree when a rule introducing an empty category (a "gap") has been applied. Thus the rightmost subtree of the analysis for *Which college did Bateman found?* has the form

```
[np(syn(...),S),[empty_np,[]]]
```

where empty_np is the rule identifier for deriving the noun phrase gap.

Example: Long Distance Dependencies

Long distance dependencies are treated entirely within the feature system, using the "gap-threading" technique described by Karttunen [15]. The idea is that in a sentence like *Who did you give the book to _?*, the filler *who* is connected to the gap position marked by "_". Recognizing this connection gives us the information we need to determine which argument of the verb is being questioned, which is necessary to interpret the sentence correctly. The connection is established by making the rules that introduce fillers send an instruction to the rest of the sentence to find a corresponding gap. In the following rule, an interrogative np is introduced as a filler, and in its sister s an np is placed on the gapsin list. Requiring the gapsout list to be empty ensures that a gap must be found if the sentence is to be grammatical:

```
syn(whq_ynq_slash_np,
    [s:[type=whq,form=tnsd,gapsin=G,gapsout=G,
       usedin=Fr,usedout=Fr],
     np:[wh=q,num=N, thirdps=T],
     s:[type=ynq,form=tnsd,
        gapsin=[np:[num=N]],gapsout=[],
        usedin=[],usedout=[g]]]).
```

Each rule "threads" the incoming gaps through all the constituents that can be missing. Thus the rule that provides the analysis of the verb phrase *give a book to the man* threads the gap features through both the np and the pp, since either might be or might contain a gap:

```
syn(vp_v_np_pp,
    [vp:[....
        gapsin=Gi,gapsout=Go,usedin=Ui,usedout=Uo],
     v: [...]
     np:[gapsin=Gi,gapsout=Gnext,usedin=Ui,usedout=Unext],
     pp:[gapsin=Gnext,gapsout=Go,usedin=Unext,usedout=Uo]]).
```

Gaps are "found" by rules like:

```
syn(empty_np,
    [np:[num=N,gapsin=[np:[num=N]|Rest],gapsout=Rest,
        usedin=Used,usedout=[g|Used]]]).
```

The gapsin feature is regarded as a stack: if we are looking for an empty np, we can find it at any point in the input, pop that request off the stack, and continue looking for one fewer empty nps.

A.3 Semantic Information

A.3.1 Logical Forms

Three levels of linguistic analysis are generated by the CLE in response to processing a sentence. These are syntactic analyses, unscoped logical forms, and scoped logical forms. The main purpose of the CLE is to produce the last of these; the other two levels can be regarded as intermediate representations built on the way to producing the scoped logical form for the sentence.

The semantic interpretation phase takes as input the syntactic analyses produced by the parser, semantic interpretation rules, lexical entries, and sortal restrictions, from which it generates unscoped logical forms. Sortal restrictions have the effect of filtering out some of the readings allowed by the semantic interpretation rules. Although quantifier scoping is part of the overall process of "semantic interpretation", we tend to reserve this term for semantic processing that happens prior to the application of the scoping algorithm.

As with syntactic analyses, semantic analyses are represented internally in the CLE as local analyses for constituents. The interpretation records for constituents encode syntactic and semantic feature values, logical form expressions, and sortal information. The logical forms shown below for complete sentences are built up by unification from these local records.

Scoped Logical Forms

The scoped logical form expressions produced by the CLE can be specified as follows:

⟨*lf-expr*⟩ → quant(⟨*quantifier*⟩,⟨*variable*⟩,⟨*restriction*⟩,⟨*body*⟩)
⟨*lf-expr*⟩ → wh(⟨*wh-sense*⟩,⟨*variable*⟩,⟨*restriction*⟩,⟨*body*⟩)
⟨*lf-exp*⟩ → [⟨*functor*⟩,⟨*argument*$_1$⟩,...,⟨*argument*$_n$⟩]
⟨*restriction*⟩ → ⟨*lf-expr*⟩
⟨*body*⟩ → ⟨*lf-expr*⟩
⟨*argument*⟩ → ⟨*lf-expr*⟩
⟨*argument*⟩ → ⟨*variable*⟩
⟨*argument*⟩ → ⟨*constant*⟩

where ⟨*quantifier*⟩, and ⟨*wh-sense*⟩ are constants. The variable terms are Prolog variables, and a constant can be any Prolog atom. It has been a deliberate design decision to try to eliminate from the CLE any dependence on the meaning of particular atomic symbols. Only quantifier scoping depends on this at all, to avoid producing logically equivalent scopings, and the necessary information about particular quantifiers and operators is isolated in a set of declarations. Otherwise, the meanings of particular atoms are provided by whatever query evaluator or other back-end is attached to the CLE.

We use the Prolog list notation for the application of a functor (or operator) to its arguments. This representation was chosen because it allows the possibility of having complex expressions in the functor position in later versions of the CLE.

To take an example, the preferred reading of the sentence *Who visited every college built by Wren?* has the logical form:

```
wh(what1,A,[agent, A],
   quant(forall,B,[and,[college1,B],
                        quant(exists,C,[event,C],
                              [build1,C,wren1,B])],
         quant(exists,D,[event,D],[visit1,D,A,B])))
```

Logical connective operators, such as and and not, which will have been introduced by interpretation rules, are treated as any other operators. In the example above, and is an operator with two arguments, and the predicate `visit1` a functor with three arguments. Quantification is represented by expressions of the form

`quant(`⟨*quantifier*⟩`,`⟨*q-variable*⟩`,`⟨*restriction*⟩`,`⟨*body*⟩`)`

where the expression ⟨*restriction*⟩ selects the objects over which the quantifier ranges. In the example, the expression

`quant(exists,C,[event,C],[build1,C,wren1,B])]`

can be read as "Is there some event `C` such that `C` is a building by Wren of `B`", or more properly, "Of the things `C` of which `[event,C]` is true, is there at least one of which `[build1,C,wren1,B])]` is true?". Expressions of the form

`wh(`⟨*wh-sense*⟩`,`⟨*wh-variable*⟩`,`⟨*restriction*⟩`,`⟨*body*⟩`)`

are similar to those for quant, except that while quant succeeds or fails, wh returns the list of things that satisfied ⟨*restriction*⟩ and that also satisfied ⟨*body*⟩ The ⟨*wh-sense*⟩ entry is the word sense of the corresponding WH-determiner (e.g., *which, what*). This is not used in the current implementation but could be used as part of the quantifier scoping algorithm.

Unscoped Logical Forms

A.3.2 Semantic Interpretation Rules

There are one or more semantic interpretation rules associated with each syntax rule. The format of a semantic rule is as follows:

sem(⟨*syntax-rule-id*⟩,
    [(⟨*logical-form*⟩,⟨*mother-category*⟩),
     ⟨*daughter-pair*$_1$⟩,...,⟨*daughter-pair*$_n$⟩]).

The logical form paired with the mother category is a template for building the semantic interpretation of the constituent analysed by the syntax rule whose identifier is ⟨*syntax-rule-id*⟩. It is a template because it typically contains variables that are unified with the semantic interpretations of the daughter constituents. Such variables appear as the left hand elements of the daughter pairs, which have the following format:

(⟨*daughter-interpretation-variable*⟩,⟨*daughter-category*⟩)

In the interpretation rule below, the variable V stands for the semantic interpretation of the first daughter (the verb constituent), and this variable appears in the logical form template associated with the mother:

```
sem(vp_v_np,
    [(quant(exists,E,[event,E],V),
      vp:[tense=T,modifiers=n,passive=n,
          subjval=Subj,gapvalsin=ValsIn,
          gapvalsout=ValsOut]),
     (V,v:[tense=T,eventvar=E,
           agentval=Subj,patientval=Np]),
     (Np,np:[gapvalsin=ValsIn,
             gapvalsout=ValsOut])]).
```

In the current version of the CLE, logical variables of the logical form language itself must be represented as Prolog variables. The event variable E in the example rule is such a variable. Although this restriction is not a consequence of the rule format as such, the present implementation depends on it in a number of ways.